Санкт-Петербургский государственный университет

Экономический факультет

Кафедра информационных систем в экономике

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

По направлению 080500- «Бизнес-Информатика»

Использование методов интеллектуального анализа данных для формирования маркетинговой стратегии компании

Выполнил:

Бакалавриант 4 курса, группы БИ-4

Шаталов Владимир Владимирович

_____/Подпись/

Научный руководитель:

(доцент, кандидат экономических наук)

Забоев Михаил Валерьевич

_____/Подпись/

Санкт-Петербург

2017

Оглавление

Введение

Последние несколько лет большое внимание как в научных публикациях, так и в СМИ уделяется интеллектуальному анализу данных. Теперь регулярно в заголовках статей и журналов фигурируют понятия: большие данные, машинное и глубокое обучение, искусственный интеллект. Если посмотреть на ученых с самым высоким индексом Хирша в области компьютерных наук, то работы первых 10 тем или иным образом связаны с разработкой методов интеллектуального анализа данных и его применения в различных областях. [1] Наиболее популярной сферой применения, не считая IT, является биоинформатика. Что касается работ, связанных с экономикой, то их заметно меньше и наиболее развитой сферой являются финансы. Однако учитывая наличие большой базы инструментов для машинного обучения, исследования в сферах, связанных с экономикой выглядят очень перспективно. Еще одной причиной так считать является стремительный рост информации, доступной для анализа. Но большая ее часть формируется пользователями в виде текста в социальных сетях, форумах и блогах. Для ее изучения раньше было принято назначать специалистов из службы поддержки или работы с клиентами. Однако, если продукт становится популярным, то упоминаний о нем становится так много, что просмотреть их все с помощью людей может стать невозможно или экономически невыгодно. Именно этим обусловлена актуальность данной работы.

Цель дипломной работы – разработка методики проведения маркетингового исследования на основе интеллектуального анализа данных на примере модели обработки отзывов о ресторанах, определяющей эмоциональный окрас отзыва. Данные были взяты из публичного датасета, предоставляемого компанией Yelp, для академических исследований. В нем содержится более 4 миллионов отзывов о различных заведениях от более чем 1 миллиона пользователей. Помимо отзывов также предоставляются подробные сведения о каждой организации, включающие в себя более 90 характеристик, а также информация о пользователях.

Объектом исследования являются компании ресторанного бизнеса. Предметом – процесс формирования маркетинговой стратегии развития ресторана с учетом информации, содержащейся в отзывах клиентов.

Были поставлены следующие задачи:

- Поиск практических примеров использования анализа отзывов с помощью машинного и глубокого обучения;

- Изучение методов интеллектуального анализа текстов;
- Подготовка данных для построения модели;
- Построение моделей;
- Анализ полученных результатов и выбор лучшей модели.

Данная работа включает в себя три главы. В первой изучается значимость интеллектуального анализа данных отзывов для компаний. Во второй рассматриваются существующие на данный момент методы и подходы к определению тональности текстов, а в третьей – изученные методы применяются на практике и сравниваются между собой.

Глава 1. Применение методов интеллектуального анализа данных в бизнесе

1.1 Предпосылки для интеллектуального анализа данных

В бизнесе же уже давно начали использовать автоматизированный анализ данных, но с ростом мощности ЭВМ, возможности их обработки вышли на новый уровень. Большие данные помогают компаниям ответить на такие вопросы как: какую цену установить на продукты и услуги, как увеличить лояльность клиентов и сократить их отток и многие другие.

Существует довольно много прикладных приложений обработки текста в бизнесе:

- Таргетированная реклама;
- Анализ тональности в маркетинге и финансах;
- Автоматическая поддержка пользователей;
- Борьба со спамом.

В данной работе будет сделан акцент на анализ тональности для маркетинговых исследований.

Большинство методов интеллектуального анализа данных создавались для работы с показателями, которые тем или иным образом описывают потребителей: доход, количество кликов, социальная группа и место жительства, пол и возраст, образование, история покупок и так далее. Для сбора и хранения всех этих данных нужна подходящая инфраструктура, создание которой требует квалифицированных специалистов. Более того, для построения математических моделей желательно иметь часть данных в динамике, а, следовательно, на их получение требуется время. Но что делать, если в компании нет подходящей инфраструктуры или она не так давно вышла на рынок. Можно собирать фокус группы и устраивать их опрос, можно запускать бета-тестирование, но иногда наиболее подходящий инструмент – изучение отзывов. Это не требует долгого сбора данных, что, несомненно, является преимуществом.

Для того, чтобы обучить модели необходимы размеченные данные. То есть если мы хотим научиться отличать положительный отзыв от негативного, то сначала надо каждому отзыву из обучающей выборки присвоить метку положительного или отрицательного настроения. Учитывая, что для достижения достаточной обобщаю-

щей способности модель необходимо обучать на большом количестве отзывов, есть два пути решения этой проблемы. Первый - использовать данные из рекомендательных сервисов, таких как Foursquare или Yelp, в которых пользователи помимо отзывов ставят оценки заведениям. У приведенных выше сервисов, существуют API для доступа к необходимой информации. Второй способ - найм асессоров, которые будут самостоятельно размечать данные. Такой способ более затратный и долгий, но его приходится использовать, если требуется анализировать специфические области, для которых не существует специальных сервисов, как в первом случае.

Можно выделить следующие причины столь частого использования машинного обучения в бизнес среде. Кроме уже упомянутого роста объема данных, сюда относится рост производительности ЭВМ, появление квалифицированных специалистов, тиражирование в СМИ.

Рост производительности ЭВМ в 1965 году предсказывал Гордон Мур. [2] Согласно первоначальной формулировке закона Мура: количество транзисторов, размещаемых на кристалле интегральной схемы, удваивается каждые 24 месяца. Со временем закон Мура претерпел некоторые изменения, но всегда сводился к быстрому росту производительности. Приведем наглядное отражение этого закона в реальном мире. Суперкомпьютер Deep Blue, созданный в 1995 году и обыгравший в 1997 году Гарри Каспарова в шахматы, имел мощность 11.4 GFLOPS, что в 220 раз меньше, мощности персональных компьютеров 2013 года с видеокартой GeForce GTX 780M. [3]. Несмотря на такой рост мощностей, оборудование для глубокого обучения все еще стоит больших денег и поэтому многие компании используют сервисы облачных вычислений. Например, компания Yelp хранит и обрабатывает данные на серверах Amazon. Ежедневно Yelp получает 1,2 терабайта новых логов и обрабатывает их, используя Hadoop. Это позволило сэкономить 55 000$ на первоначальных аппаратных расходах, а также экономить на поддержании работоспособности IT-инфраструктуры. [4] Такие сервисы облачных вычислений как AWS, Google Cloud и другие, позволяют арендовать свои мощности только на то время, когда происходят вычисления, что может быть очень удобно для обучение различных моделей. Еще одним плюсом является простота масштабирования. Если раньше создание нового центра обработки данных занимало длительный период времени, то сегодня достаточно нажать на кнопку и ваша компания получит доступ к необходимому количеству ЭВМ.

Большое внимание в последнее время уделяется обучению анализу данных. Во-первых, любой желающий может пройти онлайн обучения на платформах массового образования: Coursera, Udacity и другие. Во-вторых, у таких западных университетов, как Стэнфорд, Оксфорд, Массачусетский технологический институт материалы курсов находятся в открытом доступе. [5][6][7] В-третьих, компании, чувствуя потребность в подобного рода специалистах, организовывают с ВУЗами совместные образовательные программы. Например, Яндекс имеет свои кафедры в ВШЭ и МФТИ, а Mail.ru Group сотрудничает с МГУ и МГТУ. Все эти факторы говорят о том, что количество специалистов по анализу данных и их квалификация в ближайшее время должны вырасти.

Что касается СМИ, то за последние два года упоминания науки о данных и глубокого обучения стали встречаться все чаще. То же можно сказать и росте поисковых запросов.

Рисунок 1- Динамика упоминания в СМИ

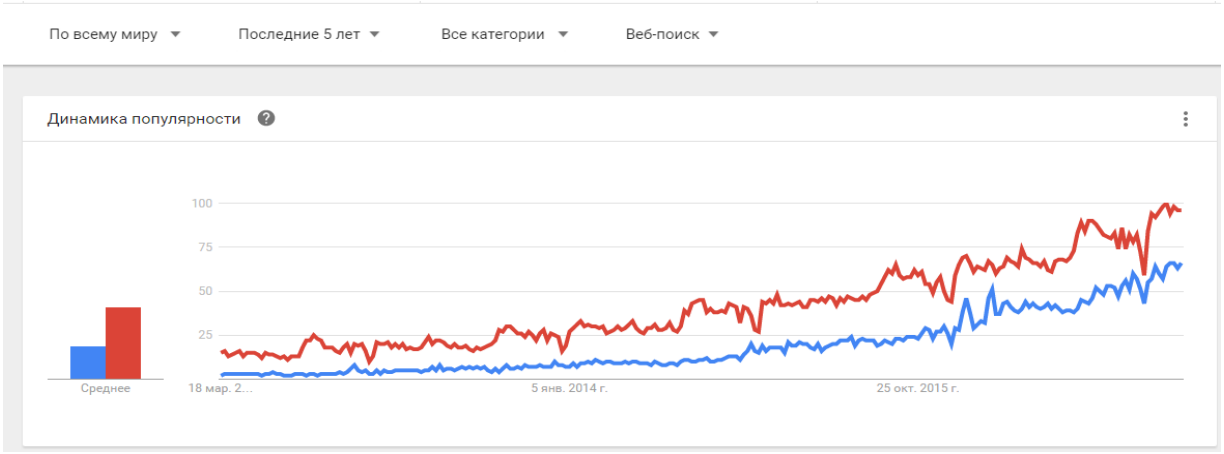


Рисунок 2 - Рост поисковых запросов

Таким образом, учитывая все перечисленные выше предпосылки, можно предположить, все большее число компаний будет использовать передовые технологии анализа данных в своей деятельности.

С другой стороны, комплексность требуемых знаний создает определенные барьеры входа. Так в компетенцию специалиста, в зависимости от выполняемых задач, могут входить:

- Определение потребностей бизнеса;
- Какие данные компании экономически выгодно собирать и хранить;
- Осуществление сбора данных, осведомленность о существующих инструментов для такого сбора данных;
- Очистка данных;
- Изучение данных: выявление закономерностей и аномалий;
- Определение аналитических методов и моделей, которые свяжут данные и потребности бизнеса;
- Создание, тестирование и проверка моделей;
- Создание визуализаций, которые облегчат анализ для конечных пользователей;
- Встраивание полученных моделей в существующие приложения или устройства.

Людей, способных совмещать выполнение всех перечисленных задач, крайне мало, поэтому широко распространена практика набора целых команд, занимающихся анализом данных. [8] В такие команды входят:

- Бизнес-аналитик – человек, хорошо понимающий бизнес-процессы и задачи. Изучает данные с помощью таблиц и визуализации данных.
- Data Scientist – специалист, хорошо разбирающийся в статистике, математике, информатике и машинном обучении.
- Разработчик – ответственный за внедрение моделей в приложение.
- Data engineer – ответственный за обеспечение хранения данных.
- DevOps инженер – отвечает за развертывание системы, организацию информационного потока.

Важные изменения произошли и в процессе управления исследованием. Если раньше этот процесс был последовательным и одна группа специалистов включалась в работу после того, как всю необходимые задачи выполнила другая группа, то сейчас, с популяризацией гибких технологий разработки, все специалисты действуют одновременно в рамках одной команды. Для совместной работы популярны ноутбуки (notebooks) и контейнеры. Ноутбуки позволяют в удобной форме обмениваться документами с исполняемым кодом, поясняющим текстом, формулами и графиками. Пример такого ноутбука представлен на рисунке 3.

Рисунок 3 - Пример ноутбука в R

Контейнеры же позволяют не заботиться о том, что у коллег не установлено совместимое окружение, например, соответствующей версии библиотеки машинного обучения. По своей сути они схожи с виртуальными машинами за тем исключением, что имеют меньший вес и быстрее создаются. Добиться слаженной работы всей команды непросто и кроме технических деталей необходимо уделять внимание управленческой составляющей процесса разработки. Так, еженедельное или ежедневное проведение стендапов позволяет сделать работу согласованной, замечать недоразумения на ранней стадии их появления. Для управления подобного рода проектами создано большое количество сервисов: Trello, Asana, Jira и другие. С помощью них каждому участнику удобно назначать задачи и следить за ходом их выполнения, чтобы всегда оставаться в курсе текущего состояния проекта. Кроме того, большин-

ство таких сервисов имеют интеграции с корпоративными мессенджерами, что делает их использование еще более удобным и эффективным, так как все уведомления попадают напрямую к пользователю и у него пропадает нужда в проверке этих уведомлений в самом сервисе. Хорошей практикой также считается регулярная демонстрация продукта коллегам, в том числе и из других отделов, как для стимулирования более активной деятельности членов команды, так и для получения фидбека.

Отличительной особенностью интеллектуального анализа данных в бизнесе является то, что большая часть программных средств распространяется бесплатно и поддерживается сообществом разработчиков. На практике это имеет следующие последствия:

- В основном, решения об использовании тех или иных программ принимают разработчики, так как нет необходимости согласовывать траты с начальством;
- Большое разнообразие справочного материала, по сравнению с платными решениями;
- Аналитики имеют возможность выбирать наиболее подходящий инструментарий для каждой конкретной задачи.

Самыми популярными решениями на данный момент являются:

- Python – высокоуровневый язык программирования, который в совокупности с библиотеками: pandas, numpy, scikit-learn и многими другими – предоставляет богатые возможности анализа данных.
- R – язык программирования, который в отличии от python изначально разрабатывался для статистических вычислений и графического представления данных.
- Scala – язык программирования, основанный на JVM1. Не имеет такого большого количества библиотек для анализа данных, как предыдущие два языка. Но зато он может использоваться в качестве языка разработки в Spark и Hadoop.
- Hadoop - это платформа с открытым исходным кодом, основанная на Java, которая поддерживает обработку и хранение чрезвычайно больших наборов данных в распределенной вычислительной среде.

[1] Виртуальная машина Java

- Spark - кластерная вычислительная платформа с открытым исходным кодом.
- Julia – современный высокоуровневый язык программирования для научных вычислений. Особенностью является его высокая производительность. Однако, так как это очень молодой язык, он еще не так популярен, как вышеперечисленные.

Несмотря на большие возможности open source решений, довольно популярны и платные решения. Например, Matlab, SPSS, SAS и т.д. Они по-прежнему остаются конкурентоспособными, но теряют долю рынка.

Рисунок 4 -Популярность продуктов для анализа данных

Как видно из графика на рисунке 4 платные продукты становятся все менее популярными и их уже довольно давно обходит R. Так как Python язык общего назначения, то сравнивать его со специализированными языками программирования было бы некорректно. Поэтому на графике представлена популярная библиотека для анализа данных в Python. Ее доля пока небольшая, однако уже в скором времени она может обогнать SPSS.

Внедрение команд, занимающихся интеллектуальным анализом данных, в рабочий процесс многих компаний не могло не привести к появлению методологий,

описывающих их деятельность. Самым известным и популярным является стандарт CRISP-DM[2], принятый еще в 1999 году. Он включает в себя следующие стадии [9]:

- Понимание потребностей бизнеса. На этом этапе определяются потребности бизнеса, а затем формулируются в терминах интеллектуального анализа данных, а также составляется план исследования.
- Изучение данных. Описание данных, исследование, проверка качества данных.
- Подготовка данных: выбор необходимых данных, очистка данных, трансформация данных, форматирование данных.
- Построение моделей. Составление набора моделей, из которых предстоит выбирать наиболее подходящую. Далее определяется, как будет оцениваться качество модели и соответствие бизнес-требованиям. После чего строятся модели и подбираются их параметры и гиперпараметры. Затем выбирается модель, которая лучше всего показала себя на тестах.
- Оценка модели. Теперь в отличии от оценки точности и скорости работы на предыдущем этапе, необходимо посмотреть, насколько полученная модель удовлетворяет бизнес требованиям. Например, позволяет ли ее использование достичь указанных в плане значений метрик. Например, если модель предназначена для прогнозирования оттока клиентов, то как она позволяет сократить отток на экспериментальной выборке. После этого в зависимости от оставшихся ресурсов и того, насколько модель хороша на практике, либо переходят к последнему этапу, либо возвращаются к первому.
- Внедрение. Подготовка автоматизированной работы полученной модели.

Даже после завершения последнего этапа предполагается поддерживание модели: отслеживание ее работы, изменение параметров модели и ее дообучение на новых данных или замена на новую.

[2] *Cross-Industry Standard Process for Data Mining*

Рисунок 5- методология CRISP-DM

Кроме рассмотренного выше стандарта, свое видение представила компания SAS. Их методология называется SEMMA и включает следующие этапы: выбор данных, исследование данных, трансформация данных, моделирование, оценка результатов и одобрение модели. Данный подход за счет меньшего количества требований позволяет ускорить разработку, но имеет значительный недостаток – не уделяется должного внимания аспектам бизнеса.

В данном параграфе были рассмотрены причины возникновения такой отрасли, как интеллектуальный анализ данных: рост объема данных, рост вычислительных мощностей, популярность в СМИ, доступность образования. За последние 10-15 лет сформировались все атрибуты, которые необходимы профессиональной сфере: свои специализированные инструменты, достаточное количество специалистов, управленческие подходы и профессиональные стандарты. Далее рассмотрим, как использование достижений интеллектуального анализа данных приносит пользу бизнесу.

1.2 Бизнес-кейсы использования машинного обучения на текстовых данных в сфере маркетинга

Ян Лекун, один из самых авторитетных ученых в области нейронных сетей, в своем интервью на вопрос о том, какие практические применения глубокого обучения[3], по его мнению, будут востребованы в ближайшие 10 лет, в первую очередь, назвал понимание естественного языка, а уже затем автомобили с автопилотом и робототехнику. При этом Лекун отмечает, что на текущий момент в обработке естественного языка нет таких успехов, как, например, в распознавании речи или изображений, и достижение удовлетворяющих наши потребности результатов в этой области - следующий большой рубеж, который предстоит преодолеть [10]

Несмотря на то, что существует большой простор для улучшения работы методов распознавания текстов, существует немало кейсов использования этих методов в деятельности компаний. Это может быть, как основная деятельность компаний, так и инструмент для достижения каких-либо целей, напрямую несвязанный с оказанием услуг и производством товаров. В качестве примера рассмотрим компанию Xeneta, предоставляющую услуги логистики другим компаниям, которые нуждаются в морских грузоперевозках. В компании существовал список потенциальных клиентов, из которого требовалось выбрать тех, кто согласится воспользоваться услугами Xeneta. На это у сотрудников уходили часы, так как требовалось изучить информацию в интернете о потенциальных клиентах. Основная задача заключалась в том, чтобы на основе описании компании понять: требуются ли ей морские грузоперевозки. В качестве модели были выбраны случайные леса, которые давали 86% точности, что принесло значительную пользу отделу продаж, ускорив процесс поиска потенциальных клиентов. [11]

Для компании ABBYY обработка естественных языков — это то, на чем основывается бизнес. Компания разрабатывает продукты со следующим функционалом:

- Преобразование изображений документов в формат, который можно редактировать на компьютере;

[3] Глубокое обучение – раздел машинного обучения, включающий в себя алгоритмы, который в своей работе помимо входных данных используют абстрактные уровни представления этих данных. Самым распространенным примеров таких алгоритмов являются нейронные сети с несколькими скрытыми слоями.

- Извлечение важной для бизнеса информации из плохо структурированных текстов;
- Классификация документов.

О потенциале сферы обработки естественного языка свидетельствуют и исследования аналитических агентств. В частности, в отчете компании MarketsandMarkets говорится о потенциале роста рынка продуктов, основанных на обработке естественного языка, с $7,6 млрд. в 2016 году до 16,07$ млрд к 2021 году. [12]. Однако стоит отметить насколько разные продукты представлены на этом рынке. Вот основные сферы применения обработки текстов:

- Поиск информации:
  - В сети Интернет;
  - Корпоративный поиск;
  - Поиск на сайтах.
- Перевод с одного языка на другой;
- Таргетированная реклама;
- Мониторинг упоминаний компании в сети Интернет и агрегация таких данных;
- Поиск синонимов и схожих по смыслу текстов;
- Автоматическое аннотирование текстов;
- Помощь при подборе персонала;
- Управление информационной безопасностью:
  - Перехват сообщений, содержащих коммерческую тайну;
  - Борьба со спамом;
- Чатботы для:
  - Поддержки пользователей;
  - Управления устройствами;
  - Заказа товаров.

И этот список не является исчерпывающим. Далее рассмотрим, как обработка естественного языка может применяться в маркетинге.

Одним из важнейших нефинансовых показателей в анализе деятельности компании является индекс удовлетворенности клиентов. Есть разные способы его измерения. Первый – провести опрос, попросив потребителей оценить удовлетво-

ренность сервисом или товаром в балльной шкале. Такой показатель обычно называют CSAT.[4]

Другой способ – расчет CSI[5]. Данный метод также требует проведения интервью непосредственно с покупателями. Однако в отличии от предыдущего подхода, в данном методе учитывается как несколько качественных показателей, так и количественные. Сначала выбираются критерии, по которым будет оцениваться индекс удовлетворенности. Потом каждого клиента просят оценить важность каждого показателя, так чтобы в сумме присвоенные веса давали 100%. Среди качественных показателей в зависимости от услуги или товара можно выбрать: сервис, вкус еды, транспортную доступность, качество товара. Пример количественных показателей: доля брака, доля вовремя доставленных товаров, уровень цен по отношению к ценам конкурентов. Сумма взвешенных значений всех критериев и дает индекс удовлетворенности клиентов. [13] Этот способ по сравнению с предыдущим позволяет оценить мнение клиентов более детализовано. Однако у обоих методов есть существенный недостаток – стоимость проведения. Из-за дороговизны подобного рода мероприятия проводятся обычно не чаще, чем раз в квартал, а то и раз в полгода, и даже раз в год. Между тем сам показатель является крайне важным и служит ключом к выявлению узких мест на всех этапах жизненного цикла производимой продукции, реинжинирингу бизнес-процессов. На рисунке 6 изображен цикл обратной связи с клиентом, который совершается каждый раз, когда компания получает мнение кли-

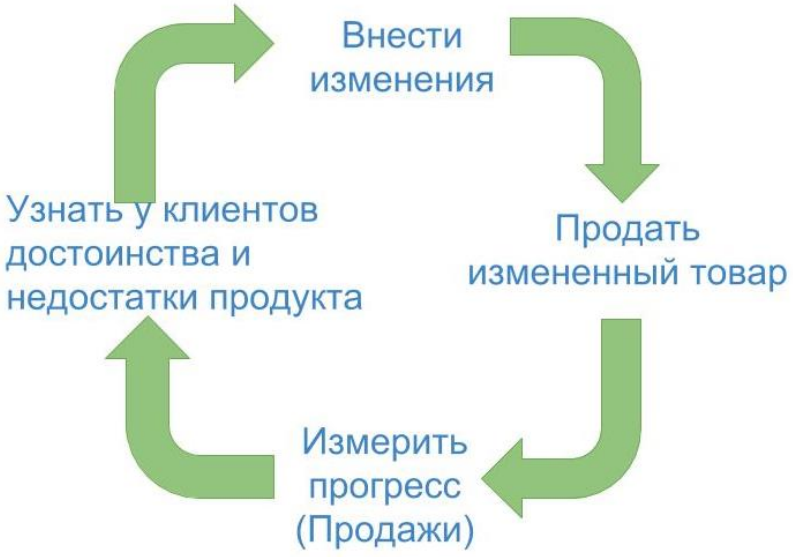


Рисунок 6 - Построение обратной связи с клиентом

[4] Customer Satisfaction
[5] Customer Satisfaction Index.

ентов о своей деятельности. Таким образом, чем чаще мы получает фидбек от пользователей, тем своевременнее вносим изменения в продукт.

Но проводить опрос – не единственный способ получить эту ценную информацию. Её также можно собрать из социальных сетей, где пользователи делятся своим мнением бесплатно, в какой-то степени более объективно и могут сообщать о таких проблемах, о которых в компании могут не подозревать. Таким образом, автоматизировав процесс сбора и оценки отзывов о компании, можно заменить опросы более дешевым и быстрым средством. Измеряя отношение пользователей к компании на основе данных, собранных в сети, мы приходим к понятию: управление онлайн репутацией. Данному аспекту маркетинга уделяется много внимания в западных странах, а само понятие довольно широкое. Оно включает в себя работу с жалобами и работу над увеличением количества позитивных упоминаний, SEO мероприятия по продвижению в поисковой выдаче материалов, выставляющих компанию в выгодном свете, но нас в данной работе будет интересовать, как обобщить информацию из сети и понять целостное отношение пользователей к компании, оценить сильные и слабые стороны компании, а также выявить ранние угрозы ее репутации и устранить их.

Стюарт Баттерфилд, создатель корпоративного мессенджера Slack - самого быстрорастущего бизнес-приложения в истории, в своем интервью Firstround отмечал важность работы с отзывами пользователей. [14]. По его словам, в его компании при поступлении жалоб или предложений, они вносятся в список на рассмотрение. Стюарт Баттерфилд уверен, что после реакции на отзывы пользователи становятся более лояльными к компании и советуют продукт друзьям и коллегам. В качестве примера таких изменений он упоминает реакцию на жалобу о большем количестве каналов. Каналы – основное место общения в Slack, которые создаются под конкретные задачи и проекты. В больших компаниях их количество быстро становится таким, что новым сотрудникам затруднительно найти к каким каналам им надо присоединиться, а какие игнорировать. В результате интерфейс был изменен таким образом, чтобы пользователь мог быстро посмотреть описание канала и его участников. На начало 2015 года в Slack было 18 человек, ответственных за круглосуточный мониторинг отзывов, которых ежемесячно накапливалось около 18 000. С тех пор количество пользователей увеличилось более чем в 8 раз и компания встала перед выбором: каким-то образом автоматизировать процесс или продолжать увеличивать отдел по работе с отзывами. [15]

Компании должны отслеживать свою репутацию, так как ее построение требует длительного времени, и ее потеря может дорого обернуться. В США рассчитывается показатель ACSI[6], оценивающий множество фирм из разных областей. Для его расчета используется как интервьюирование, так и количественные методы. Для полученных данных была обнаружена взаимосвязь с такими показателями как: ROI[7], коэффициент Тобина[8], денежный поток, волатильность денежного потока и другие. [16]

Отчасти это связано с тем, что, зная отношение покупателей к нашим действиям, мы можем принять решение о том, что предпринимать дальше. Рассмотрим пример, который это наглядно демонстрирует. Так, компания Nabisco, выпускающая печенье Oreo, в рамках своей маркетинговой кампании решила на время изменить классический белый цвет крема. Эта акция вызвала серьезный шум социальных сетях, и было решено для оценки успешности акции использовать анализ тональности. Выяснилось, что 80,9 % отзывов были положительными и всего 19,1 % негативными. [17] В результате такие акции еще не раз повторялись и были приурочены к праздникам и событиям, значимых в каких-то регионах.

На сегодняшний день технологии, позволяющие реализовать описанный выше потенциал обработки естественного языка, доступны практически всем компаниям. Ниже перечислим некоторые из сервисов, предоставляющие услуги обработки текстовой информации.

Во-первых, для полноценного анализа необходимо собрать данные из разных источников и привести их к единому виду. Такие компании как Gnip, Datasift, Xignite, Diffbot, Kimono и Connotate предоставляют такие услуги.[18] Все из этих компаний, кроме Diffbot, предлагают гибкое ценообразование для каждой компании. Стоимость использования Diffbot начинается от 299$ в месяц для небольшой команды.

Во-вторых, полученные данные необходимо каким-то образом обработать. Так можно воспользоваться услугами по поиску лидеров мнений, измерению эффективности маркетинговой компании по продвижению бренда, визуализации упоминаний компании, выявления трендов на ранней стадии.

---

[6] American Customer Satisfaction Index.
[7] Return On Onvestment – показатель окупаемости инвестиций
[8] Коэффициент Тобина – показывает долгосрочную стоимость фирмы.

Следующие компании специализуются на конкретных вышеперечисленных пунктах: Meltwater, Cision, Sysomos, Luminoso Dashboard, NewsWhip. Перечисленные сервисы также устанавливают цену индивидуально с каждой компанией.

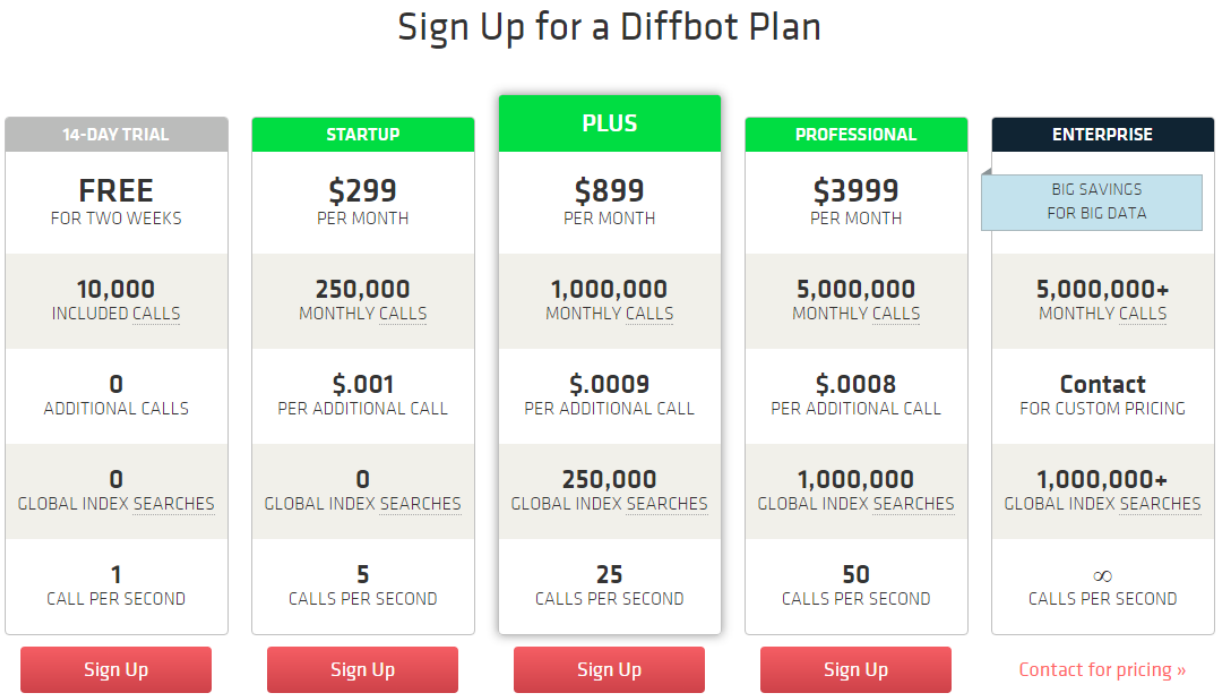


Рисунок 7- Тарифный план Diffbot

Еще одна группа продуктов, которая позволяет в реальном времени получать информацию о клиентах и других компаниях, представлена сервисами LexisNexis News Company Research и InsideView for Sales. Эти компании собирают данные о новых контрактах конкурентов, о новых продуктах и отзывах о них, о судебных решениях, найме персонала и многом другом.

Во втором параграфе первой главы мы рассмотрели примеры практического применения технологий обработки текстовых данных, а также выяснили почему важно измерять уровень удовлетворенности клиентов. В частности, знание того, как клиенты относятся к компании, позволяет выявить проблемы в продуктах и услугах, оценить направление изменения спроса, а также, увеличивая количество лояльных пользователей, высоко оценивающих деятельность компании, получить бесплатную рекламу и эффективную рекламу в виде сарафанного радио. Также приведены примеры сервисов, которые позволяют автоматизировать и внедрить обработку текстовой информации в маркетинговую деятельность компании.

Глава 2. Методы машинного обучения для обработки естественного языка

2.1 Предварительная обработка текста

В данной главе будут рассмотрены походы к предварительной обработке текста, способы представления текстов в машиночитаемом виде, а также методы, которые применяются для анализа тональности. Алгоритмы разделены на две категории:

- Основанные на деревьях решений: деревья решений, случайный лес, градиентный бустинг.
- Нейронные сети: сети прямого распространения, рекуррентные сети, сверточные сети.

Кроме того, приведены библиотеки языка Python, с помощью которых описанная теория была применена на практике.

При обработке естественного языка данные могут поступать из разных источников и быть зашумлены. Поэтому перед построением моделей всегда проводят предварительную обработку текста, включающую два этапа: токенизацию и нормализацию.

Цель токенизации – разбиение предложения на отдельные слова. При этом важно удалить символы, которые вносят шум и снижают предсказательную способность модели: теги, знаки препинания, смайлы и т.п. Также имеет смысл приводить слова к нижнему регистру, так как иначе модель не будет понимать, что слово, написанное с заглавной буквы в начале предложения и позже встреченное это же слово в середине предложения, являются одной сущностью. На практике знаки препинания часто заменяются на пробелы, а числа на специальное слово, обозначающее число. Существуют слова, которые могут встретится в любом предложении, независимо от его смысла. К таким словам относятся союзы, частицы или слова из предметной области. Их тоже принято удалять. При этом стоит пробовать разные варианты токенизации и учитывать предметную область, для которой составляется модель. Например, при распознавании эмоций важную роль могут играть восклицательные и вопросительные знаки, регистр или цифры. Также некоторые слова в разных контекстах могут приобретать разный смысл. Рассмотрим два предложения: “В данной работе применяется метод деревья классификации” и “Власти Санкт-Петербурга выделили средства на посадку деревьев в Таврическом саду”. В первом предложении говорится о методе машинного обучения, а во втором о растении. В таких случаях

одним из выходов является применение биграмм, триграмм и т.д. То есть в одну сущность выделяются сразу несколько рядом стоящих слов. Однако всегда остается вероятность, что слова стоят слишком далеко друг от друга. К тому же не стоит забывать о том, что тексты пишутся людьми и для людей, поэтому, если из предыдущего контекста понятен смысл слова, то слова, которые помогали определить этот смысл, могут отсутствовать в предложении. В данной работе для токенизации использовались встроенные методы библиотеки Keras, библиотека re для регулярных выражений и стоп-слова из библиотеки NLTK.

Нормализация – приведение слов к их начальной форме. Например, слова “деревья”, “деревьев” и “дерева” заменяются на “дерево”. Это особенно актуально, когда у нас не так много данных и нам следует сократить число используемых признаков. А так как в данном случае признаки формируются на основе слов, то уменьшая количество уникальных слов, мы уменьшим и число признаков. Есть два основных способа нормализации текстов: стемминг и лемматизация.

Используя стемминг, мы по определенным правилам удаляем из слов окончания и суффиксы. У такого подхода есть очевидные недостатки. Например, он не учитывает, что некоторые слова при изменении формы меняются целиком: “шел”, ”иду”, ”пойду”. Также некоторые слова могут быть урезаны слишком сильно, вследствие чего мы получим одну сущность из совершенно разных слов.

Лемматизация более сложный, но более распространенный, ввиду своей эффективности метод. В рамках данного подхода используется уже готовый словарь, на основании которого слова приводятся к нормальной форме. Если же слово новое и в словаре не встречается, то по определенному алгоритму выбирается способ изменения данного слова. Такой подход лишен недостатков стемминга, но при этом преобразования текста занимает больше времени. В библиотеке NLTK существует несколько реализаций стемминга и лемматизации, которые приведены в третьей главе.

После токенизации и нормализации данных можно приступать к извлечению признаков, по которым будет происходить предсказание.

2.2 Извлечение признаков для классификации

В этом параграфе описаны разные по сложности подходы к извлечению признаков: от составления мешка слов до отображения слов в векторное пространство, а

также способы уменьшения числа признаков без существенной потери в точности, что позволяет получить существенный выигрыш в производительности.

Начнем с рассмотрения одного из самых простых и популярных способов – мешка слов[9]. Данный подход позволяет извлекать признаки, которые можно подавать на вход как нейронным сетям, так и алгоритмам, основанным на построении деревьев. Так как мы уже разбили предложения на токены, то вполне можем составить единый словарь всех токенов. Затем для каждого текста из нашей коллекции мы можем посчитать, какое количество раз в нем встретился каждый токен. В результате получится матрица, в которой строкам соответствуют тексты, столбцам – токены, а на пересечении i-ой строки и j-ого столбца находится количество токена j в тексте i. Такая матрица и называется мешком слов. Рассмотрим два предложения:" деревья классификации и деревья принятия решений – один алгоритм" и "мы будем говорить деревья классификации". Для этих предложений получим следующий словарь: деревья, классификации, и, принятия, решений, -, один, алгоритм, мы, будем, говорить. Тогда полученная матрица будет иметь следующий вид.

Деревья	классификации	и	принятия	решений	-	один	алгоритм	мы	будем	говорить
2	1	1	1	1	1	1	1	0	0	0
1	1	0	0	0	0	0	0	1	1	1

Рисунок 8 - Пример мешка слов

Несмотря на то, что мы в процессе токенизации удаляем стоп-слова, это вовсе не означает, что в тексте не останется слов содержащих мало информации для дальнейшей классификации. Так, если слово встречается в большом количестве документов[10], то скорее всего оно менее важно, чем слово, встречающееся в небольшой группе документы. По этой причине часто оказывается полезно каким-либо образом взвесить значения, полученные при создании мешка слов. Одним из таких способов получил название TF-IDF[11]. [19] На первом этапе рассчитывается, как часто встречается слово в документе.

$$tf(t,d) = \frac{n_t}{\sum_k n_k} \quad (1)$$

[9] Далее Bag Of Words

[10] В нашем случае документ равнозначно отзыв. Предполагается, что в одном документе может содержаться несколько отзывов.

[11] Term Frequency – Inverse Document Frequency

Где, n_t- число вхождений слова t в документ d, а в знаменателе стоит количество всех слов в документе.

Далее вычисляется обратная частота встречаемости слова в документах:

$$idf(t) = \log \frac{1 + n_d}{1 + df(d,t)} + 1 \quad (2)$$

Где, n_d – общее число документов, а $df(d,t)$ - количество документов, содержащих слово t.

Далее считается произведение

$$v = tf(t,d) * idf(t), \quad (3)$$

после чего в каждом документе полученные значения нормируются по следующей формуле:

$$v_{norm} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}} \quad (4)$$

Необходимо заметить, что подход TF-IDF означает лишь, что взвешивание значений из мешка слов будет осуществляться, исходя из принципа, что вес слова пропорционален количеству употребления этого слова в документе, и обратно пропорционален частоте употребления слова в других документах из корпуса. [20] Сами же формулы для расчета могут отличаться. В данной работе они приведены из реализации алгоритма в библиотеке Scikit-learn для Python.

Серьезным недостатком мешка слов является то, что с ростом числа документов растет и объем словаря, что приводит к экспоненциальному росту сложности вычислений. [21] Кроме увеличения времени вычислений, увеличивается и объем памяти, необходимый для хранения полученных матриц. На данный момент, это не всегда может быть выполнено эффективно, так как реализации большинства алгоритмов не поддерживают работу с разреженными матрицами.

Чтобы решить эти проблемы, часто в ущерб точности работы моделей используются методы сокращения размерности. Рассмотрим два наиболее популярных метода. Первый – усеченное сингулярное разложение[12]. Пусть полученный мешок

[12] Далее SVD

слов представлен матрицей X порядка $m \times n$, где m – количество документов, а n – размер словаря. Сингулярным разложением будет разложение следующего вида: $X = UDV^T$, где D – матрица размера $m \times n$, на главной диагонали которой лежат сингулярные числа, а все остальные элементы равны нулю. U размера $m \times m$ и V размера $n \times n$ ортогональные матрицы. Мы можем попытаться приблизить матрицу X матрицей $\tilde{X}$ меньшего ранга. Для этого мы должны найти такую матрицу $\tilde{X}$, чтобы разность $X - \tilde{X}$ по норме Фробениуса была минимальной. Для ранга k такую матрицу можно получить из усеченного сингулярного разложения.

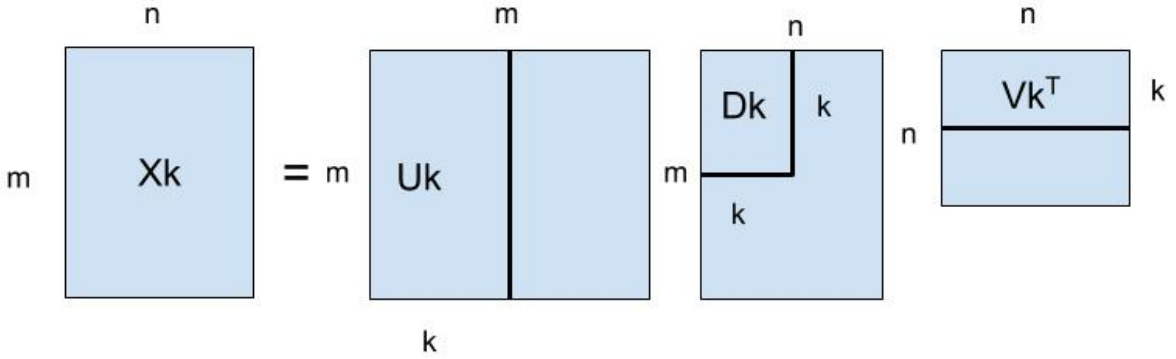


Рисунок 9 - Усеченное сингулярное разложение

На рисунке 9 наглядно показано, как получается усеченное SVD. Из матрицы U возьмем первые k столбцов, из матрицы D - квадрат размером $k \times k$, соответствующий самым большим сингулярным числам, из матрицы V^T – первые k строк. Таким образом, получаем $X_k = U_k D_k V_k^T$. Оказывается, что матрица X_k будет наилучшим приближением матрицы X по норме Фробениуса. [22] Тогда в мы можем заменить матрицу X на матрицу U_k, сократив тем самым количество признаков с n до k. [23] Данная процедура реализована в библиотеке Scikit-learn. К ее преимуществам можно отнести и то, что она умеет обрабатывать разреженные матрицы, что ускоряет операцию.

Еще один известный способ сократить количество признаков без существенной потери качества – метод главных компонент.[13] В рамках этого подхода ищется такое ортогональное подпространство признаков, что при проецировании элементов из первоначального пространства на это подпространство сумма дисперсий всех наблюдений будет максимальна. Максимизация дисперсии означает, что размерность пространства снижается так, чтобы терялось как можно меньше информации. А так как вектора в новом базисе ортогональны, то мы избавляемся от сильно корре-

[13] Далее PCA

лированных признаков. Уменьшение размерности начинается с того, что все признаки центрируются. Затем рассчитывается матрица ковариации. Далее можно искать векторы, при проекции на которые дисперсия была бы максимальна. Из отношения Рэлея следует, что наибольшая доля дисперсии сохраняется при проецировании на собственные векторы, соответствующие наибольшим собственным числам. [24][25] Иными словами, искомые главные компоненты и являются собственными векторами. Чтобы получить проекции на новое пространство достаточно умножить транспонированную матрицу из k собственных векторов на исходную матрицу признаков, где k – требуемое количество признаков. Отметим, что из-за того, что данные центрируются, то алгоритм не поддерживает работу с разреженными матрицами и занимает больше времени по сравнению с усеченным SVD.

Описанные выше методы хорошо подходят, когда используются алгоритмы, основанные на деревьях и нейронные сети прямого распространения. Для рекуррентных сетей используется другое преобразование. Из всех токенов создается словарь, так что каждому токену соответствует уникальный номер. При этом соблюдается закономерность – чем чаще встречается слово, тем меньше его порядковый номер. Затем все токены заменяются на свои номера. Чтобы все документы имели одинаковую длину либо удаляется необходимое количество токенов, либо недостающие значения заполняются нулями.

Пожалуй, самым лучший на сегодняшний день способ извлечения признаков из текстов – это отображение слов в векторное пространство с помощью нейронных сетей таким образом, чтобы близкие по смыслу слова располагались как можно ближе друг к другу. Компании Google и Facebook предоставили в открытый доступ свои алгоритмы Word2Wec и FastText. [26][27]. Это достаточно современные методы: в 2013 году Томас Миколов представил статью, в которой описывал Word2Wec, а в 2016 он же совместно с другими сотрудниками Facebook Research опубликовал статью о FastText. [28][29] Их может использовать любой желающий, в том числе в коммерческих целях. На рисунке 10 показана главная особенность такого подхода. На нем представлена проекция пространства, полученного в результате Word2Wec на две первые главные компоненты. Как видим, если мы прибавим к вектору Германия вектор Польша и вычтем вектор Берлин, то получим вектор очень близкий к Варшаве. Однако эта модель не позволяет полностью представить смысл слов в векторном пространстве. Так, например, при умножении какого-либо элемента на -1 мы не получим антоним этого слова, как того можно было ожидать. В дан-

ной работе мы будем пользоваться реализацией Word2Wec в библиотеке genism. Если в наличии нет достаточного объема текстов, чтобы обучить модель, то можно воспользоваться готовыми векторными представлениями слов. Facebook не так давно выложил в своем репозитории на Github такие модели на 90 языках, включая русский. [30]

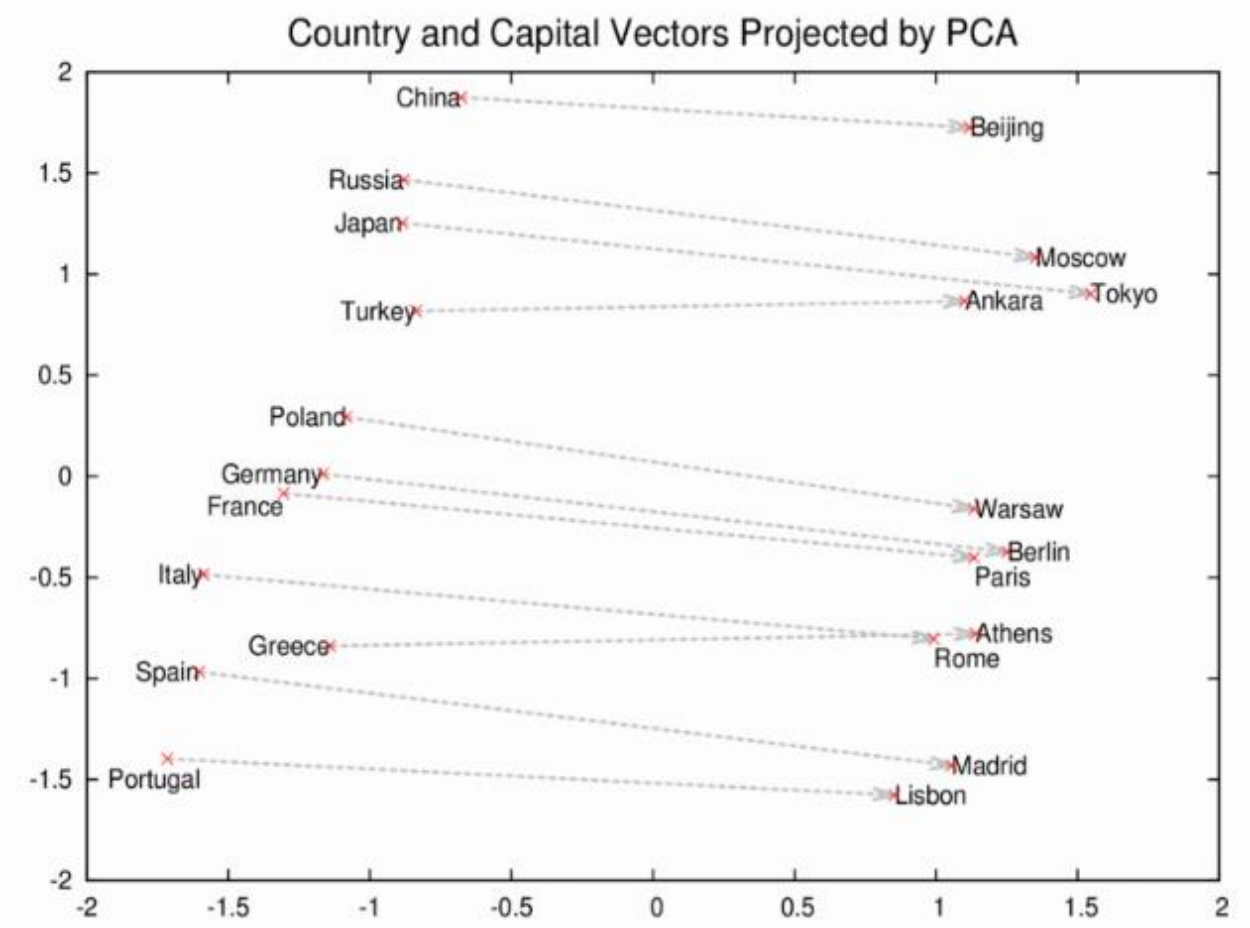


Рисунок 10 - Визуализация Word2Wec

В основе этих методов лежит идея, что слова со схожим смыслом часто встречаются в контексте с одними и теми же словами. Существует две разновидности модели Word2Wec: CBOW[14] Model и Skip-Gram Model. В первом случае по контексту предсказывается слово, во втором наоборот. Рассмотрим, как устроена Skip-Gram Model и как получить искомые векторы. Сначала выбирается длина так называемого окна. Иными словами, как далеко могут находится слова, чтобы все еще считаться близкими друг к другу. Пробегая таким окном по всему тексту, мы будем

[14] Continuous Bag of Words

Рисунок 11 - пример окна в Word2Wec

получать пары слов: рассматриваемое слово, которое будет подаваться на вход нейронной сети, и соседнее, которое необходимо будет предсказать. Поясним на примере приведенного ранее предложения: “ Мы будем говорить деревья классификации”. Предположим, что длина окна равна пяти, то есть два предыдущих и два последующих слова будут считаться находящимися рядом. Как в этом случается будут выбираться пары слов, показано на рисунке 11. Для каждого слова применяется one-hot кодирование: слово представляется в виде вектора, размерность которого равна величине словаря, на месте, соответствующем позиции слова в словаре располагается единица, а все остальные элементы равны нулю. Таким образом, первое слово из каждой пары будет подаваться на вход, а второе на выход. Архитектура используемой нейронной сети представлена на рисунке 12.

Нейроны скрытого слоя не имеют функции активации и нужны для расчёта весов. Другими словами, скрытый слой представляет собой матрицу весов, количество строк в которой равно размеру словаря, а количество столбцов - размерности нового пространства. В результате обучения модели эта матрица и будет представлением слов в векторном пространстве. В нейронах выходного слоя используется функция активации Softmax. Данная функция активации имеет следующую формулу:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{к=1}^{К} e^{z_k}}, \quad (5)$$

где z – число, получившееся после умножения входного вектора на вектор весов, соответствующий слову с номером i. Значения каждого выходного нейрона больше нуля, а в сумме дают единицу. То есть каждый i-ый нейрон дает нам представление о вероятности, с которой с i-ое слово встретится рядом с входным словом. Получается, что если два слова часто встречаются вместе, то нейронная сеть подберет такие веса, которые будут давать схожее распределение на выходе. А значит такие слова будут располагаться рядом в нашем векторном пространстве. [31]

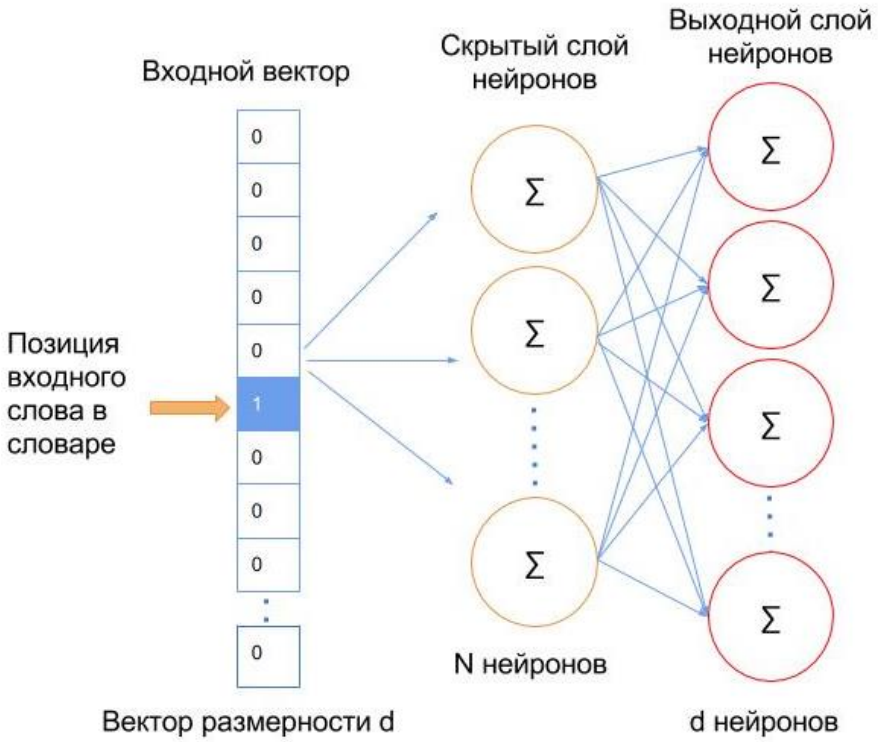


Рисунок 12 - Архитектура Skip-Gram Model

Отличие FastText от Word2Wec заключается в том, что в этом алгоритме слова делятся на составные части, что обеспечивает лучшую работу для морфологически богатых языков. [32] Перед тем как закончить разговор о такого рода нейронных сетях стоит отметить одну из их важнейших особенностей – они не требуют предварительной разметки данных. Это одни из немногих очень эффективных алгоритмов обучения без учителя. И в ближайшем будущем нас может ожидать множество новых методов обучения без учителя, так как размечать такие большие объемы данных, которые появляются каждый день очень затратно.

В параграфе 2.2 показано, как можно получить признаковое описание текстов, чтобы в дальнейшем на основе полученных признаков обучать модели. В следую-

щем разделе рассмотрим модели, в основе которых лежат деревья классификации и как они применяются для работы с текстами.

2.3 Методы, основанные на деревьях

Начнем с описания деревьев классификации. Они изучаются уже довольно давно и относятся к той группе алгоритмов, для которых было предложено особенно много уточнений и изменений. Поэтому можно говорить о целом семействе алгоритмов, основывающихся на одних и тех же принципах, но отличающихся в некоторых деталях. В sklearn, который используется в данной работе, используется оптимизированный алгоритм CART. [33]

Модель представляет собой бинарное дерево, в узлах которого содержится какое-то множество объектов из первоначальной выборки и значение атрибута, по которому происходит разбиение этого множества на два подмножества. Алгоритм построения является рекурсивной процедурой. На первом шаге в корень помещается вся выборка, затем выбирается значение одного из атрибутов, которое наилучшим образом делит выборку на два множества, после чего в зависимости от значения атрибута конкретного экземпляра, он отправляется в левого или правого потомка. На следующем шаге происходит выбор наилучшего значения атрибута уже для потомков. Эта процедура продолжается до тех пор, пока не будет выполнено хотя бы одно из следующих условий:

- Число наблюдений в узле стало меньше наименьшего допустимого значения;
- Достигнута максимально возможная глубина дерева;
- В случае раскола модель не улучшится на минимально необходимое значение.

Последний пункт проще всего понять на примере регрессионных моделей: если на i-том шаге R^2 увеличится менее, чем на минимально допустимое значение, то этот узел стоит сделать листом. Самыми популярными критериями для разбиения узлов являются: информационная энтропия и коэффициент Джини. Деревья классификации, пожалуй, самый слабый алгоритм из тех, которые мы рассмотрим, склонный к переобучению. Но при этом он очень хорошо интерпретируем. В рамках нашей задачи это означает, что после построения дерева мы можем получить понятные правила, которые будут объяснять, что именно не нравится и наоборот нравится пользователям в нашем сервисе. Также этот алгоритм работает гораздо быстрее слу-

чайных лесов и градиентного бустинга, что позволяет перебрать большое количество параметров и гиперпараметров при его построении, чтобы выбрать лучшую модель.

Однако дерево само по себе довольно слабый классификатор. Поэтому на практике в основном композиции деревьев.

В 1785 году Николя де Кондорсе в своей работе “ Эссе о применении анализа к вероятности решений, принятых большинством голосов” сформулировал теорему о жюри присяжных. Смысл ее в том, что, если каждый член жюри присяжных принимает решение независимо от других и вероятность того, что это решение верное, больше чем 0.5, то вероятность правильного вердикта жюри возрастает с ростом количества числа присяжных и стремится к 1. В противном случае, если вероятность правильного решения каждого из членов жюри меньше 0.5, то вероятность правильного решения присяжными в целом монотонно уменьшается и стремится к нулю с увеличением количества присяжных. [34] Несмотря на то, что данная теорема была сформулирована в рамках политической науки, она также нашла отражение в предложенном Лео Брейманом и Аделем Катлером методе машинного обучения, получившим название случайный лес. В этом методе используется множество деревьев, а конечный результат получается на основе усреднения голосов. Случайные леса умеют решать задачи классификации, регрессии и даже кластеризации. Но так как третьей главе стоит задача определения тональности, то будем рассматривать данный алгоритм в контексте классификации. Изначально имеем множество всех наблюдений. Из этого множества сформируем подмножества на основе выборок с повторениями. Эта процедура получила название бутстрэп. Далее на каждом из подмножеств обучим дерево классификации. Но решающие правила в деревьях будем получать, производя поиск не по всем признакам, а по выборке из них. После того, как необходимое количество деревьев построено, все предсказания усредняются.

Такая процедура позволяет сократить дисперсию, за счет чего достигается прирост в точности классификации. Базовые модели получаются довольно разнородными, так как строятся на разных множествах и используют разные признаки для классификации. Благодаря этому, по задумке создателей, они должны компенсировать ошибки друг друга, а итоговый классификатор не должен переобучаться. Но как мы увидим в главе 3 – это не всегда так.

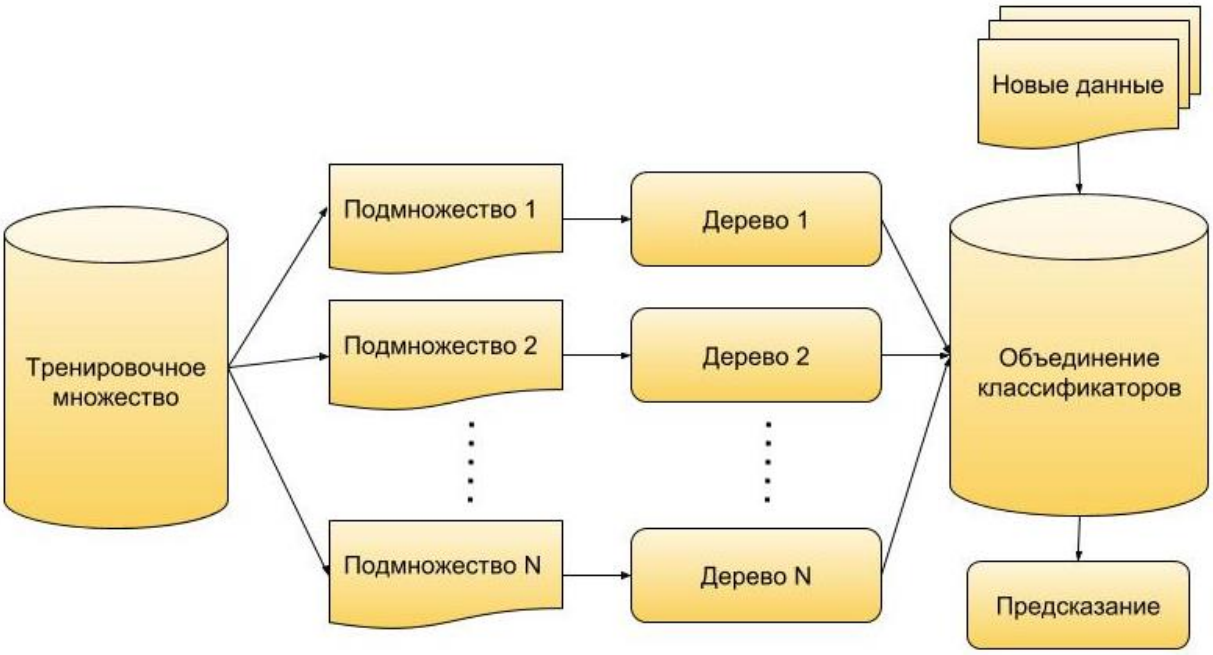


Рисунок 13 - Построение случайного леса

Если все-таки наблюдается переобучение, то можно изменить алгоритм и для каждого признака выбирать порог, по которому разделяется множество не из всех возможных вариантов, а из некоторого подмножества. Такой подход получил название экстремально случайные леса. [35] В отличии от деревьев классификации, мы не получим информации о том, как именно предикторы влияют на результат работы алгоритма, но все еще можем выбрать самые важные атрибуты. Также случайные леса менее склонны к переобучению. Главным недостатком в предстоящей задаче будет то, что при построении каждого дерева выбирается небольшое количество признаков, что не лучшим образом сказывается на качестве классификации в случае сильно разреженных матриц. Также из-за большого числа признаков обучение занимает довольно долгое время, что, впрочем, частично компенсируется возможностью параллельного построения деревьев на разных ядрах компьютера.

Последний алгоритм, который мы рассмотрим в данном параграфе, - градиентный бустинг. В отличии от случайного леса, где усредняются решения всех классификаторов, в градиентном бустинге они суммируются. Базовые алгоритмы строятся один за другим так, что каждая следующая модель исправляет ошибки уже построенной композиции. В качестве базовых алгоритмов чаще всего берут неглубокие деревья классификации. На первом шаге выбирается совсем простой классификатор, а далее происходит итеративное улучшение. Предположим, что уже обучен N-1 ал-

горитм. Обозначим композицию следующим образом: $a_{N-1}(x) = \sum_{n=1}^{N-1} b_n(x)$. Стоит задача подобрать модель $b_N(x)$, которая бы при добавлении в композицию, увеличила бы точность классификации. Это можно записать через функцию потерь L(y,z), где y – вектор с фактическими классами, а z – вектор с предсказаниями. В качестве функции потерь можно взять, например, среднеквадратичную ошибку или кросс-энтропию.

$$\sum_{i=1}^{l} L(y_i, a_{N-1}(x_i) + b(x_i)) \underset{b}{\rightarrow} min \tag{6}$$

Классификатор $b(x_i)$ возвращает вектор значений $s = (s_1, \dots, s_l)$. Таким образом мы можем переписать задачу минимизации.

$$F(s) = \sum_{i=1}^{l} L(y_i, a_{N-1}(x_i) + s_i) \underset{b}{\rightarrow} min \tag{7}$$

То есть необходимо найти вектор s, который будет минимизировать функцию $F(s)$. Для этого достаточно вычислить антиградиент функции $F(s)$ по всем точкам из обучающего множества. Таким образом,

$$s = -\nabla F = \begin{pmatrix} -L'_z(y_1, a_{N-1}(x_1)) \\ \dots \\ -L'_z(y_l, a_{N-1}(x_l)) \end{pmatrix} \tag{8}$$

После вычисления антиградиента, мы можем перейти к подбору оптимальной модели.

$$b_N(x) = argmin_b \frac{1}{l} \sum_{i=1}^{l} (b(x_i) - s_i)^2 \tag{9}$$

После того, как алгоритм $b_N(x)$ найден, он прибавляется к текущей композиции с небольшим весом, чтобы избежать переобучения.

$$a_N(x) = a_{N-1}(x) - \eta b_N(x) \tag{10}$$

Стоит отметить, что, как и в случае со случайными лесами, обучение деревьев на разных подмножествах дает лучшие результаты. Для борьбы с переобучением в градиентном бустинге наиболее важны два гиперпараметра: количество деревьев и длин шага η. Основным недостатком при обучении является то, что нельзя произво-

дить параллельные вычисления моделей, так как все они строятся последовательно, а не параллельно, как в случайных лесах.

В этом параграфе мы рассмотрели алгоритмы, в основе которых лежат деревья классификации, их преимущества и недостатки. Далее перейдем к нейронным сетям, которые в последнее время все чаще используются для обработки текстовых данных.

2.4 Модели классификации на основе искусственных нейронных сетей

Сразу оговоримся, что в данном параграфе рассмотрены виды нейронных сетей для решения задач обучения с учителем, то есть построения алгоритма, обучаемого на парах объект-ответ, который с удовлетворяющей нас точностью бы определял ответы для новых объектов. Иными словами, ограничимся задачами регрессии и классификации. Кроме того, с помощью нейронный сетей можно также решать задачи кластеризации, уменьшения размерности, визуализации данных и другие.

Начнем с рассмотрения сетей прямого распространения. Они получили свое названия из-за того, что в них существуют только связи, идущие от входных нейронов к выходным, и отсутствуют обратные связи.

Нейронная сеть состоит из входного слоя, скрытых слоев (не всегда), выходного слоя, а также связей между нейронами соседних слоев. Уникальность многослойных нейронных сетей заключается в том, что они способны самостоятельно генерировать признаки в скрытых слоях. Например, подавая на вход изображение, на скрытых слоях нейронные сети выделяют характерные черты этих изображений: от кривых до черт лица. На входной слой подается некоторый вектор $X = (x_i, \dots, x_n)$, каждый элемент которого, умноженный на определенный вес w_i,передается на следующий слой в соответствии с установленными между нейронами связями. На следующем слое к взвешенной сумме входных значений и константы применяется некоторая функция активации $f(x)$. В качестве функции активации можно брать линейную функцию, гиперболический тангенс, сигмоиду, ReLU[15] и другие функции. В случае, когда в сети есть скрытые слои, тенденцией стало использовать ReLU. Эта функция имеет следующий вид

$$f(Wx + b) = \max(0, Wx + b) \tag{11}$$

[15] Rectified Linear Unit

С одной стороны, данная функция не является всюду дифференцируемой, что затрудняет обучение, так как оно основано на вычислении градиента. С другой стороны, она вычисляется проще, чем сигмоида и гиперболический тангенс. Также у нее за счет линейности отсутствует насыщение, что позволяет избежать затухания градиента и значительно увеличить скорость обучения. Результат вычисления функции активации является выходным значением нейрона. Далее аналогичные действия продолжаются для последующих слоев. Таким образом, нейронную сеть можно представить в виде композиции функций: $f(x) = f^{(n)}(f^{(n-1)}(\dots\left(f^{(1)}(x)\right)\dots)$, где f^n – функция активации выходного слоя. В выходном слое может быть от одного нейрона в случае регрессии до сколь угодно большого количества в случае с задачей многоклассовой классификации. Выходные функции могут быть такими же, как и в скрытых слоях. В нашем случае, когда предстоит классифицировать несколько классов, хорошим выбором будет описанный в параграфе 2.2 Softmax. С ее помощью формируется вероятностное распределение по всем классам. [36] Стоит отметить, что перечислены далеко не все функции активации, а лишь самые популярные.

Важным аспектом при проектировании нейронных сетей является выбор функции потерь, которая определяет, как будет считаться ошибка. К счастью, в нейронных сетях используются такие же функции потерь, как и в других методах машинного обучения. Для регрессии наиболее популярный выбор среднеквадратичная ошибка:

$$L(Y, \hat{Y}) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{12}$$

Для классификации – перекрестная энтропия:

$$L\left(Y, \hat{Y}\right) = -\frac{1}{n}\sum_{i=1}^{n}\left[Y_i log\hat{Y}_i + (1 - Y_i)\log(1 - \hat{Y}_i)\right] \tag{13}$$

Часто для того, чтобы избежать переобучения к функции потерь добавляют сумму абсолютных значений весов, используемых в модели, умноженных на коэффициент регуляризации, или сумму квадратов весов, умноженных на коэффициент регуляризации. Тогда говорят, что используется L_1-регуляризация и L_2-регуляризация. Другой способ борьбы с переобучением получил название дропаут. [36] В рамках данного подхода в процессе обучения связи с заданной вероятностью

исключаются из модели. Таким образом, обучается некоторый ансамбль моделей и затем результаты усредняются.

Для того, чтобы подобрать веса модели применяется метод обратного распространения ошибки. Как понятно из названия, процесс изменения весов начинается с выходного слоя, где в соответствии с функцией потерь вычисляется ошибка. Далее необходимо изменить веса таким образом, чтобы в следующий раз значение ошибки уменьшилось. Обозначим функцию потерь $L(Y, \hat{Y})$, функцию активации i-го нейрона - $F_i(x)$, скорость обучения - η, вес связи от i-го нейрона к j-ому - $w_{i,j}$, линейная комбинация выходов предыдущих нейронов для i-го нейрона - $x_{i,j}$, выходное значение i-го нейрона - σ_i. Тогда веса, с которыми участвуют нейроны последнего скрытого слоя изменяются по следующей формуле:

$$w_{i,j}^{new} = w_{i,j} - \eta * \frac{\partial L_j}{\partial \hat{Y}} * \frac{\partial F_j}{\partial x_{j,i}} * \sigma_j \tag{14}$$

Таким образом, получаем формулу градиентного спуска, где ошибка вычисляется как $\frac{\partial L_j}{\partial \hat{Y}} * \frac{\partial F_j}{\partial x_{j,i}}$. Чтобы продолжить изменять значение весов в нейронной сети, надо вычислить ошибку на скрытом слое. Для этого уже известная ошибка нейрона следующего слоя умножается на новый вес связи нейрона, для которого ищется ошибка, и нейрона, для которого ошибка известна. Таким образом, изменяются все веса в нейронной сети от выходного слоя до входного. [37]

Градиентный спуск в чистом виде редко используются для поиска минимума функции потерь. Как правило, он изменяется определенным образом, чтобы ускорить процесс обучения и избежать локальных минимумов. [38] Обновление весов для классического градиентного спуска можно записать следующим образом:

$$\Delta\theta = -\eta \nabla_\theta L(\theta) \tag{15}$$

$$\theta = \theta + \Delta\theta \tag{16}$$

Где θ – параметры сети, $L(\theta)$- функция потерь, η - скорость обучения.

Чтобы быстрее достичь минимума, применяется так называемый импульс. Теперь при изменении весов учитываются не только значение градиента в настоя-

щий момент времени, но и в предыдущие. Формула для коррекции весов приобретает следующий вид:

$$v_t = \gamma v_{t-1} - \eta \nabla_\theta L(\theta) \tag{17}$$

$$\theta = \theta - v_t \tag{18}$$

Изменяя параметр γ, мы можем изменять влияние предыдущих значений градиента. Также мы можем считать градиент не от текущих значений параметров, а измененных с учетом направления движения. То есть вместо $\nabla_\theta L(\theta)$ будем считать $\nabla_\theta L(\theta - \gamma v_{t-1})$. Это позволит быстрее двигаться, если производная увеличивается в той стороне, куда мы двигаемся, и наоборот.

Также существуют методы адаптивного градиента. В них отпадает необходимость со временем уменьшать скорость обучения. Один из них – RMSProp[16]. Он призван исправить следующий недостаток градиентного спуска. В задачах с несбалансированными выборками желательно отличать обрабатывается редкое наблюдения или же нет. Иначе велика вероятность, что из-за большого количества наблюдений одного класса, сеть настроит свои веса в большей степени на их определение. Для каждого веса теперь хранится история его обновлений.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{19}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{E[g^2]_t + \varepsilon} g_t \tag{20}$$

$E[g^2]_t$ - скользящее среднее, g_t^2 – квадрат градиента, ε – очень маленькая константа, которая позволяет избежать случая деления на ноль.

Существуют алгоритмы, сочетающие в себе предыдущие идеи. Например, Adam[17]. Формула для v_t такая же, как и в случае градиентного спуска с импульсом, а для $E[g^2]_t$ как в RMSProp, с той лишь разницей, что коэффициенты γ могут отличаться. При этом в начале обучения мы искусственно увеличиваем значения этих параметров: v_t до 10 первых итераций, $E[g^2]_t$ до 1000.

[16] Root Mean Square Propagation
[17] Adaptive Moment Estimation

$$\hat{v}_t = \frac{v_t}{1 - \gamma_1^t} \tag{21}$$

$$\widehat{E[g^2]}_t = \frac{E[g^2]_t}{1 - \gamma_2^t} \tag{22}$$

Веса обновляются по следующей формуле:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{E[g^2]}_t + \varepsilon}} \hat{v}_t \tag{23}$$

Отметим, что существуют другие методы оптимизации, однако они практически не применимы на больших объемах данных.

Еще одной особенностью оптимизации является то, что в большинстве случаев для вычисления градиента используют небольшую подвыборку данных размером от 32 наблюдений до 1024, которую также называют батчем. Но наблюдения выбираются не случайно, а последовательно, в этом принципиальное отличие от бутстрэпа. Это делается для того, чтобы увеличить скорость обучения, так как оценка градиента по небольшой выборке все еще хорошо указывает направление движения. Размер батча выбирается из следующих соображений:

- Большие батчи обеспечивают более точную оценку градиента.
- Большие батчи требуют больше памяти, что важно так, как в основном они обрабатываются параллельно.
- При вычислениях на GPU быстрее обрабатываются батчи, размер которых является степенью двойки.

Обобщая выше сказанное, становится очевидным, что подбор достаточно хорошей конфигурации нейронной сети занимает много времени, так как предстоит выбор большого числа характеристик: количество нейронов и количество слоев, функции активации для каждого слоя, функция потерь, способ оптимизации, способ борьбы с переобучением, размер батчей. Также необходимо согласовать с бизнес задачами метрику для оценки качества работы модели. Это может быть просто доля правильных ответов, как в нашем случае, так и более сложные варианты: ROC-AUC, F-мера, R^2, MSE. [39] Кроме того необходимо выбрать количество эпох, то есть количество полных проходов по всей выборке.

Сверточные нейронные – особый вид нейронных сетей прямого распространения, получивших свое название из-за особого слоя, на котором вместо операции умножения происходит свертка. На вход в таких сетях подаются уже не только векторы, но и матрицы. В таких матрицах могут храниться, например, временные ряды, изображения, тексты. Под сверткой в нейронных сетях понимается операция суммирования элементов, получившихся в результате произведения Адамара для матриц A и B. Это операция показана на рисунке 14.

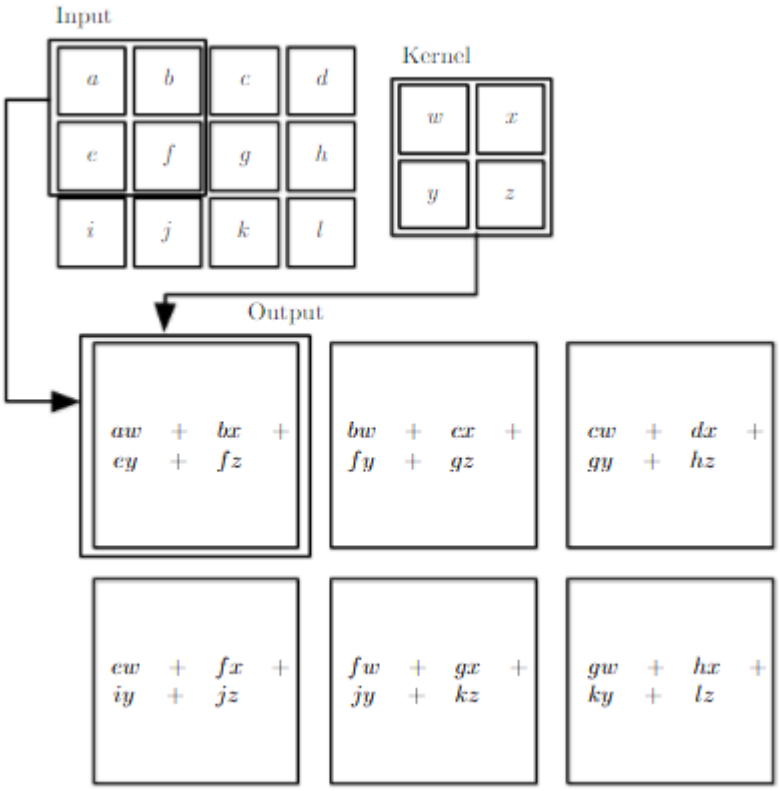


Рисунок 14 - Свертка в нейронных сетях

Матрица A - некоторая область входной матрицы. Мы получаем такие матрицы двигаясь по всей входной матрице с некоторым шагом. Матрица B также называется фильтром или ядром и отвечает за генерацию конкретной абстракции. Например, в случае обработки изображений, она может отвечать за выделение окружностей. Таких фильтров может быть довольно много на одном уровне и все равно это приводит к увеличению скорости обучения, так как если в полносвязных слоях каждый нейрон был связан с каждым, то при свертке одни и те же фильтры с небольшим количеством весов используются для всех входных данных. Изначально матрицы B заполняются случайными значениями, а веса подбираются все тем же методом обратного распространения ошибки. Есть у данной операции и свои минусы. Так, в модель добавляется три новых гиперпараметра. Во-первых, предстоит подобрать ко-

личество фильтров. Во-вторых, с каким шагом будем двигаться по входной матрице, чтобы получать матрицы A. В-третьих, иногда полезно искусственно расширить матрицу нулями, чтобы сохранить больше информации на первых этапах. Матрицы полученные в результате свертки называются картами признаков.

После слоя свертки, как правило, находится слой субдискретизации или пулинг. Он предназначен для уменьшения размерности карт признаков. К группам элементов из получившихся после свертки матриц применятся операция, которая позволяет получить из всего набора значений одно. При этом важно, что эти группы элементов не пересекаются. Чаще всего выбирается максимальное число, реже рассчитывается среднее. Обычно используется несколько чередующихся слоев свертки и пулинга. В ходе движения к выходу сети уменьшается размерность матриц, описывающих входные данные, вплоть до скаляра, но увеличивается их количество. Перед выходным слоем принято также добавлять полносвязные слои. Среди плюсов сверточных нейронных сетей можно отметить: меньшее, по сравнению с классическими сетями прямого распространения, количество весов, широкие возможности параллельных вычислений. Среди недостатков – добавление новых слоев, а значит новые гиперпараметры, которые предстоит подбирать.

К сожалению, сети прямого распространения не способны учитывать порядок слов в предложениях. Для борьбы с этим недостатком используются рекуррентные нейронные сети. Они работают не только с входными данными, но также часть нейронов может передавать информацию себе на вход. Иными словами, в рекуррентных сетях разрешены циклы. На рисунке 15 приведен пример рекуррентной нейронной сети с одним нейронном в скрытом слое.

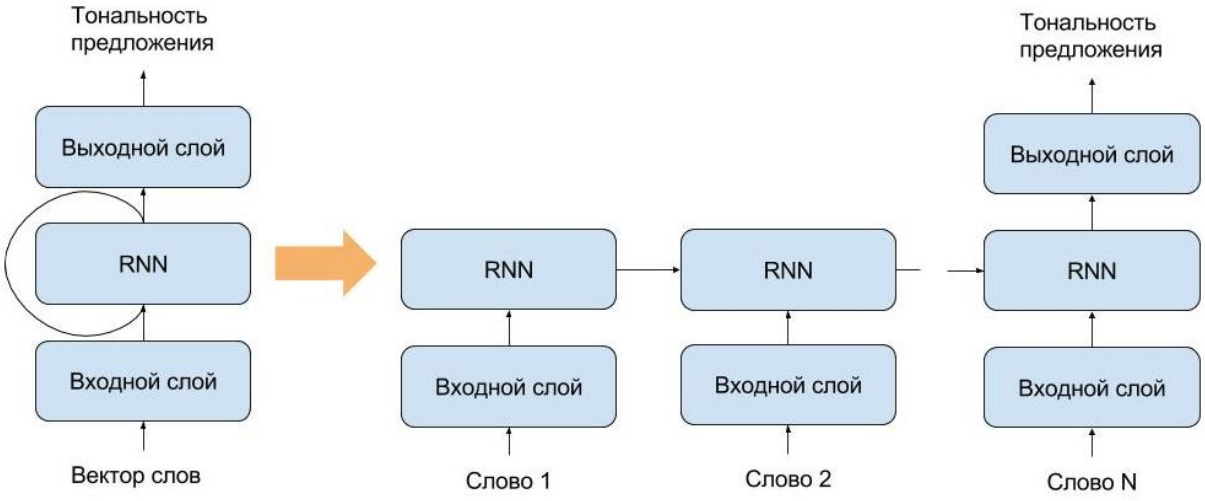


Рисунок 15 - Рекуррентная сеть

Существуют разные типы нейронов, которые могут подавать информацию себе на вход. Одними из самых распространенных являются LSTM нейроны. Такие нейроны содержат целых пять функций активации: сигмоиду, которая отвечает за то, какую часть информации необходимо забыть, сигмоду и гиперболический тангенс для обновления текущего состояния нейрона, сигмоду и гиперболический тангенс, определяющих, какая информация подается на выход. [40]

В этой главе были рассмотрены методы предварительной обработки текстов, а также алгоритмы, которые могут быть использованы для определения тональности высказываний. В следующей главе рассмотрим, как описанная теория реализуется с помощью языка Python и его библиотек.

Глава 3. Практическое использование методов интеллектуального анализа данных для обработки информации на естественном языке

3.1 Использование библиотек языков программирования для обработки текстов

Сам по себе Python является языком общего назначения, и может быть использован для широкого круга задач. Среди других возможных для использования в данной работе языков рассматривались R и Matlab. Python был выбран по двум причинам:

- Так как этот язык предназначен не только для научных вычислений, то в нем гораздо больше инструментов для работы с текстами;
- Большое число библиотек для построения нейронных сетей, разработанных как университетами, так и корпорациями. Многие из них не представлены ни в R, ни в Matlab.

Python является интерпретируемым языком программирования, а следовательно довольно медленным, поэтому для работы с матрицами принято использовать библиотеку NumPy, которая позволяет добиться скорости работы сравнимой с Matlab. Для того, чтобы с таблицами было удобно работать, как в специально разработанном для статистических исследований языке R, будем использовать библиотеку Pandas, которая является надстройкой над NumPy. Многие методы, которыми будем пользоваться реализованы в библиотеке Scikit-learn. Среди них деревья классификации, случайные и экстремально случайные леса, градиентный бустинг, метод главных компонент, усеченное сингулярное разложение, создание мешка слов и TF-IDF. Для визуализации будем использовать библиотеки Matplotlib и Seaborn. Для лемматизации, токенизации и стемминга - Natural Language Toolkit. Чтобы обучить Word2Vec будем использовать Gensim. Для построения нейронных сетей был выбран Keras, который предлагает довольно высокоуровневое описание моделей. Данная библиотека работает поверх других библиотек: Theano или Tensorflow. Мы в качестве бэкенда будем использовать Tensorflow, так как в последнее время она активно развивается компанией Google, а также по умолчанию проводит вычисления на GPU.

3.2 Описание исходных данных для практической реализации методов

Мы будем использовать данные, которые предоставляет компания Yelp[18], для академических исследований.

Набор данных содержит исчерпывающую информацию о 144 тысячах компаний в США, Канаде, Великобритании и Германии. Общее количество отзывов по всем предприятиям составляет более 4 миллионов. Данные содержатся в 6 таблицах:

- Бизнес: название компании и ID, адрес, время работы, категория заведения, название, город, рейтинг, уровень шума, атмосфера, есть ли доставка и многие другие характеристики. Всего 99 параметров.
- Чекины[19]: время и ID компании.
- Отзывы: ID отзыва, пользователя и заведения. Текст отзыва и выставленная оценка, дата, а также оценки полезности данного отзыва.
- Короткие отзывы: ID заведения и пользователя, дата, количество лайков и текст.
- Пользователи: дата регистрации, ID пользователя, количество друзей, друзья и т.д.
- Фотографии: ID фотографии, ID заведения, описание и фотография.

Для наших целей достаточно таблиц с данными о компаниях и отзывах. Части этих таблиц представлены в Приложении 1. После конвертирования данных из формата JSON в формат CSV и их загрузки в наше рабочее пространство найдем города с наибольшим количеством ресторанов. Этими городами оказались Лас-Вегас, Финикс, Монреаль, Шарлотт, Питтсбург и Эдинбург. Код данной и последующих операций представлен в приложении 2. В качестве объекта для проверки гипотез возьмем Питтсбург. Сформируем таблицы с отзывами и соответствующими им рейтингами, после чего разделим данные на обучающую и тестовую выборки, где размер тестового множества будет составлять 20% от исходного. Распределение классов на тестовом и обучающем множестве совпадают. При этом отзывов с оценкой 4 и 5 примерно в четыре раза больше, чем с оценкой 1 и 2, и в два раза больше, чем с оценкой 3. Стоит учитывать, что такой дисбаланс в классах может сказаться на работе модели. Для обучения нейронной сети заменим значение с оценкой на вектор

[18] Yelp – сайт, на котором пользователи могут оставлять отзывы о местных бизнесах, например, парикмахерские, стоматологии, рестораны и т.д.

[19] Сообщение своего месторасположения

длиной 5, где на месте соответствующем оценке стоит единица, а остальные элементы равны нулю.

3.2 Построение нейросетевых моделей для распознавания тональности текста

Для первой модели создадим мешок слов с TF-IDF, удаляя стоп-слова. Размер словаря зададим равным 5000. Меньший размер приводит к тому, что для части отзывов не находится ни одного слова в словаре. Проверим какие функции активации, методы оптимизации и функции потерь лучше подходят для данной задачи. Данные нейронные сети содержат 800 нейронов в скрытом слое и 5 в выходном. Все модели будем обучать по три эпохи, а размер батча равен 200. Результаты приведены в таблице 1.

Таблица 1 - сравнения нейронных сетей с разными функциям потерь и алгоритмами оптимизации

Функция активации	Оптимизатор	Функция потерь	Точность на обучающей выборке	Точность на тестовой выборке
ReLU, Софтмакс	Adam	Перекрестная энтропия	68,77%	58, 13%
ReLU, Софтмакс	Nadam	Перекрестная энтропия	74,19%	57,04%
ReLU, Софтмакс	SGD	MSE	32,96%	30,01%
ReLU, Софтмакс	RMSprop	Перекрестная энтропия	66,87%	58,15%
ReLU, Софтмакс	Adam	Кусочно-линейная	51,2%	52,74%
ReLU, Добавил еще один скрытый слой с 400 нейронами и функцией актива-	Adam	Перекрестная энтропия	75,96%	56,24%

ции tanh, softmax				

Как видим, полученные модели являются переобученными, несмотря на то, что обучались всего три эпохи. Далее будем использовать Adam в качестве алгоритма оптимизации и перекрестную энтропию, как функцию потерь. Выбор данного алгоритма оптимизации обусловлен не только лучшим результатом, но и тем, что нет необходимости подбирать скорость обучения.

Если посмотреть на отзывы внимательно, видно, что там остались служебные символы. Почистим данные, удалив стоп-слова и пунктуацию, кроме восклицательных знаков, а также переведем слова в нижний регистр. Сначала построим две модели на мешке слов без взвешивания слов. Будем использовать такую же архитектуру, как и ранее, но с двумя эпохами, так как предыдущие модели переобучились. Результаты на тестовых множествах для моделей, для которых использовались батчи по 200 и 500 наблюдений, 58,99% и 58,71% соответственно. При этом модели все еще сильно переобучались. Применим TF-IDF и построим новые модели. Так как с меньшим размером батча результаты получились немного лучше, то будем его уменьшать. Среди моделей, обученных на данных с такими преобразованиями лучшей стала та, которая обучалась всего одну эпоху, а размер батча был равен 30. Ее точность на обучающем множестве 57,57%, а на тестовом 59,81% . Таким образом, удалось не только увеличить точность, но и избавиться от переобучения. Далее посмотрим, как изменение числа нейронов скажется на точности. Результаты представлены в таблице 2.

Таблица 2- Влияние архитектуры и размера батча на точность

Архитектура	Размер батчей	Точность на обучающей выборке	Точность на тестовой выборке
1 скрытый слой (800 нейронов)	32	57,59%	60,19%
1 скрытый слой (400 нейронов)	32	65,09%	59,1%
1 скрытый слой (400 нейронов)	512	62,2%	60,67%
1 скрытый слой	1024	63,54%	60,19%

(400 нейронов)			
2 скрытых слоя (600, 400 нейронов)	512	64,62%	59,27%
2 скрытых слоя (512, 128 нейронов) и 2 дропаута (вероятность обнуления 0.7)	256	62,71%	59, 72%

Проведенная очистка текста лишь немного позволила увеличить точность. Теперь сделаем мешок слов с биграммами и продолжим экспериментировать с моделями. Начнем с уже использованной конфигурации с 400 нейронами на скрытом слое. Обучать модель будем две эпохи, а размер батча 512. Точность на обучающем и тестовом множествах равна 67,54% и 60, 57%. Как видим модель по-прежнему сильно переобучается. Чтобы избавиться от переобучения добавим дропаут, который будет случайным образом отключать 70% нейронов скрытого слоя во время обучения. Чтобы не потерять в точности на срытом слое теперь будем использовать 600 нейронов. Для такой модели точности равны 61,3% и 61,24%. Как видим дропаут действительно позволяет бороться с переобучением. Добавим скрытый слой с 32 нейронами и функцией активации tanh. Все также будем использовать дропаут после каждого скрытого слоя. Из-за того, что дропаут убирает довольно много нейронов, потратим на обучение 4 эпохи. Это лишь незначительно увеличивает точность до 61, 39 на тестовом множестве. Сделаем TF-IDF и обучим модель с одним скрытым слоем с 748 нейронами и дропаутом. Точность на тестовом множестве выросла до 61, 85%. Далее сделаем две отдельные ветви. На обеих будет 256 нейронов на скрытом слое и допаут с коэффициентом 0.5, но разные функции активации: ReLU и гиперболический тангенс. Затем объединим их и подадим на выходной слой с софтмаксом. Результат получился сопоставимым с предыдущей моделью: 61,81% на тестовой выборке. Посмотрим, как скажется увеличение словаря на точности. Создадим мешок слов с TF-IDF, в котором будет 10 000 слов. Построим простую модель с одним скрытым слоем, содержащим 600 нейронов, и дропаутом с коэффициентом 0,7. После трех эпох обучения на тренировочном множестве точность стала 66,45%, а на тестовом – 62,47%. Несмотря на переобучение, данная модель дала лучший резуль-

тат, из чего можно сделать вывод, что размер словаря важен в нашей задаче. Но при этом это в несколько раз увеличивает время обучения.

Посмотрим, как покажут себя рекуррентные сети в данной задаче. Начнем с того, что сделаем токенизацию, удалив при этом стоп-слова, и составим словарь слов. Далее в соответствии с полученным словарем закодируем каждый отзыв. Так как отзывы имеют разную длину, то ограничим длину 490 словами и будет добавлять нули в начало, если слов меньше, и удалить первые слова, если их больше. Первый слой рекуррентной сети будет сопоставлять каждому элементу входной последовательности вектор определенного размера. На первом этапе ограничим размерность 32 элементами. Далее идет рекуррентный слой со 100 нейронами долгой кратковременной памятью. Завершается все, как и в предыдущих конфигурациях, выходным слоем с 5 нейронами и функцией активации софтмакс. Обучив такую модель на протяжении трех эпох точность на тестовой выборке оказалась равна 59,13%, что уступает предыдущим моделям. При этом время обучения каждой эпохи более чем в 20 раз превышает время обучения одной эпохи самой большой сети прямого распространения. Добавление дропаута не улучшает результат. Ошибки модели показаны на рисунке 16. Видно, что в основном она путает соседние оценки.

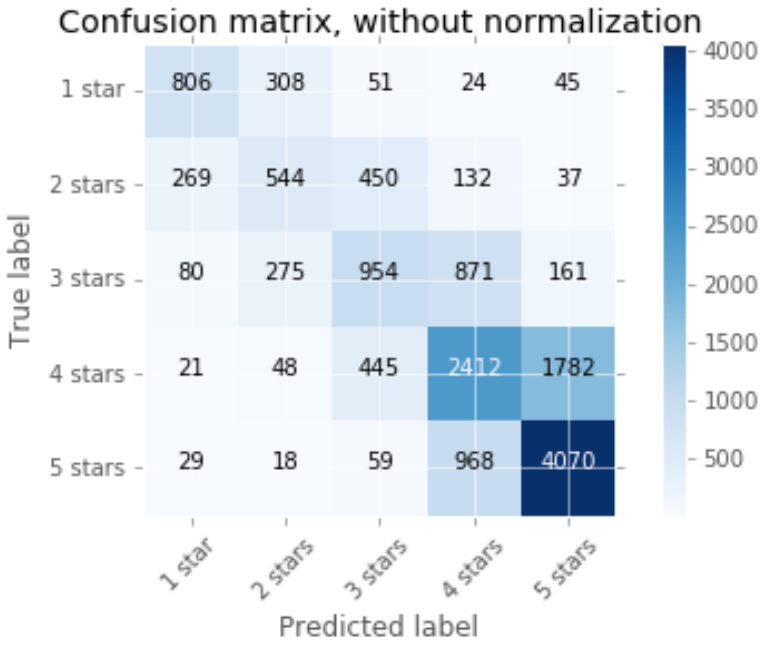


Рисунок 16 - Матрица ошибок рекуррентной нейронной сети для несбалансированной выборки

Также гораздо лучше предсказываются отзывы с 4 и 5 классами, что является следствием несбалансированной выборки. Так рейтинг 5 определяется с точностью

79%, 4 – 51%, 3 – 41%, 2- 38%, 1- 65%. Такие результаты обусловлены тем, что пользователи перечисляют как достоинства, так и недостатки заведения и определить, что для них является более важным может быть затруднительно даже для человека. Попробуем обрабатывать текст иначе, удалив из списка стоп-слов: слова очень, нет, ни, слишком и только. Возможно, это поможет отличать соседние классы. Обучив новую нейронную сеть, удалось добиться совсем незначительного улучшения результатов. Теперь точность составила 59,52%. Модель стала лучше предсказывать оценки 5, 4 и 2, но при этом точность для 3 упала на 8%. Добавление еще одного слоя с LSTM нейронами увеличивает точность до 58,76%. Но также значительно увеличивает время обучения. Если вместо LSTM нейронов использовать GRU[20] нейроны, то точность для модели с одним скрытым слоем из 100 нейронов увеличивается до 61%. Также отметим, что такая модель не является переобученной.

Посмотрим теперь, какие изменения в матрице ошибок произойдут, если сделать выборку сбалансированной. Так как у нас нет недостатка в отзывах, создадим выборку, в которой каждый класс будет представлен 20 000 наблюдений, из которых 18 000 тысяч отправим в обучающее множество и 2 000 в тестовое. По-новому будем чистить текст. Теперь еще не будем убирать вопросительные знаки, знак доллара и проценты, также заменим все числа на слово, которое будет обозначать число. Создадим мешок слов с биграммами и TF-IDF размером в 5 000 слов. Обучим на нем нейронную сеть прямого распространения с 748 нейронами на скрытом слое и дропаутом с коэффициентом 0,7. Точность на тестовой выборке 61,66%. Но гораздо интереснее то, как предсказывается каждый отдельный класс. Это показано на рисунке 17. Видим, что теперь классы определяются более равномерно. При этом по-прежнему точность выше для крайних оценок. Однако описанные выше проблемы, нейронная сеть все еще решить не может. С размерами словаря в 2 500, 10 000 слов и 15 000 слов точность на тестовой выборке равна 60,44%, 62,03 % и 61,86% соответственно. Сделаем мешки слов с TF-IDF с триграммами на 10 000 и 5 000 слов. Это позволяет еще немного увеличить точность: 62,13% и 61,35%. Сделаем лемматизацию и стемминг и обучим модели на мешке слов из биграмм размером 5 000 с TF-IDF. Точность на тестовых множествах 61,35 и 61,00% соответственно.

[20] Gated recurrent units – тип нейронов с обратной связью, представленный в 2014 году в статье J.Chung, C.Gulcehre, K.H. Cho, Y. Bengio Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.

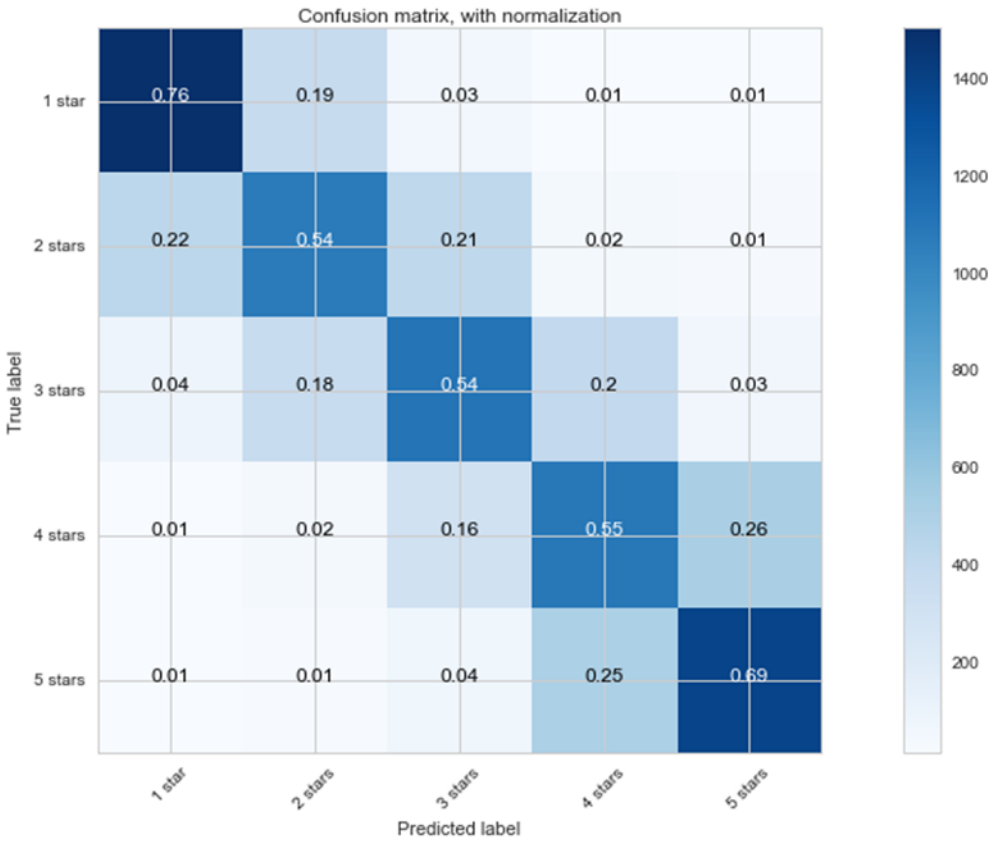


Рисунок 17- Матрица ошибок на сбалансированной выборке

В данном подразделе были рассмотрены различные конфигурации нейронных сетей для распознавания тональности текстов. Далее построим модели, основанные на деревьях и сравним результаты.

3.3 Построение моделей, основанных на деревьях

Все модели в данном разделе обучались на отзывах, из которых были удалены стоп-слова, пунктуация кроме восклицательных и вопросительных знаков, символов доллара и процентов, а также все числа заменены на слово, обозначающее число. Использовался мешок слов с биграммами размером 5 000 слов с TF-IDF. Для выбора оптимальной модели в каждом классе мы будем использовать случайный поиск по сетке. Мы можем позволить себе перебирать большое количество параметров, так как библиотека sklearn позволяет работать с разреженными матрицами и распараллеливать процесс вычислений. Для деревьев классификации будем перебирать модели со следующими параметрами:

- Критерий разбиения: индекс Джини, энтропия
- Максимальная глубина: от 4 до 61

- Максимальное число признаков для узла, среди которых выбирается лучшее разбиение: нет ограничений, $\sqrt{n}$
- Минимальное число наблюдений в узле: 1,5,10,15,20,25,30,40,50
- Минимальная доля примеси в узле: 0.1, 0.05, 0.01, 0.005, 0.001, 0.0001, 0.00001, 0.000001, 0

Чтобы лучше оценить модели будем использовать кросс-валидацию с тремя фолдами. При этом будем сохранять одинаковое распределение классов на обучающих и тестовых множествах. Тестовое множество будет составлять 20% от исходного. Лучшая модель получилась со следующей конфигурацией: индекс джини, максимальная глубина 57, без ограничения на максимальное число атрибутов в узле, минимальное число наблюдений в узле 30, минимальная доля примеси в узле 10^{-7}. Ее точность составила всего 44,79%, что значительно меньше, чем у нейронных сетей.

Построение случайных лесов требует гораздо больше времени, поэтому сначала подберем количество деревьев, а потом для этого количество подберем долю от всех признаков, по которым будет производиться разбиение узла. Результаты подбора параметров представлены на рисунках 18 и 19.

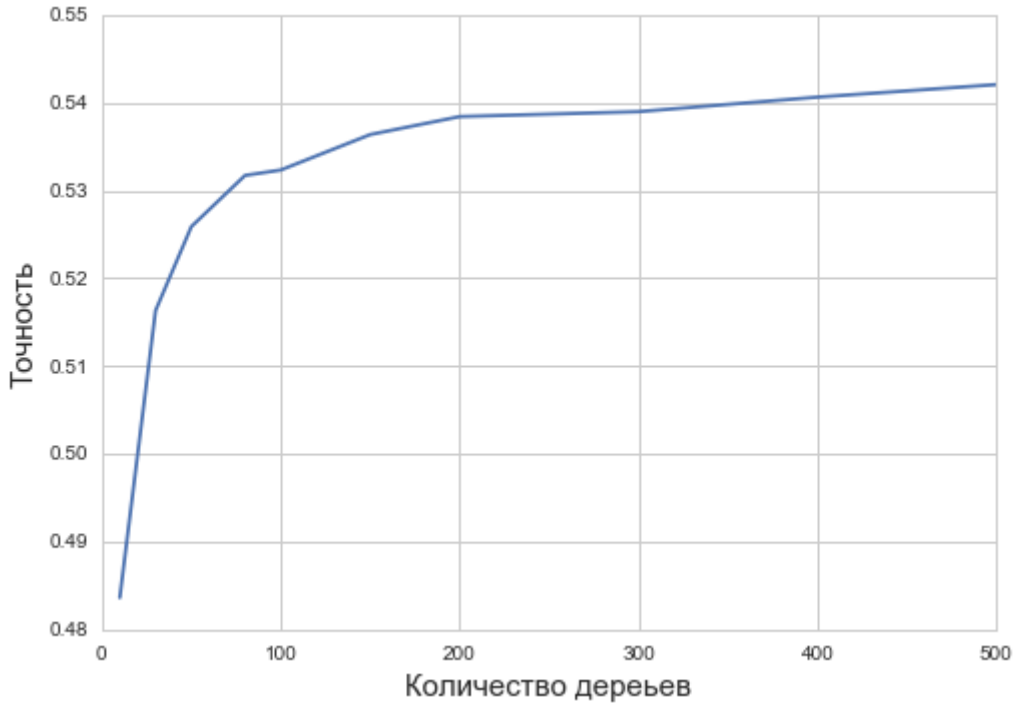


Рисунок 18 - зависимость точности от размера леса

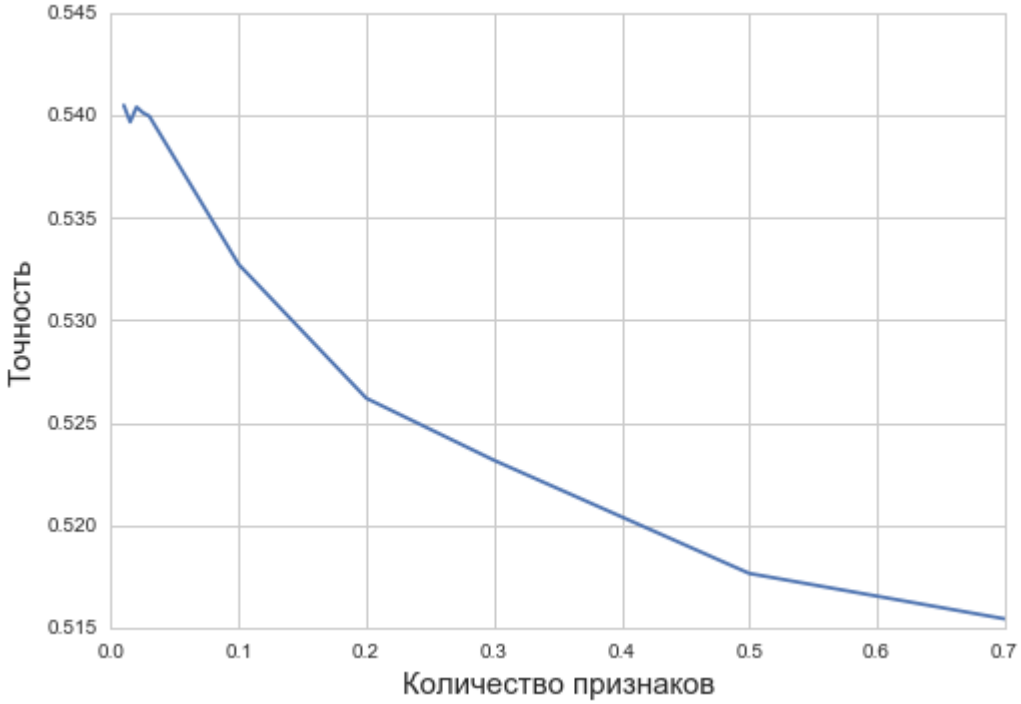


Рисунок 19 - зависимость точности от доли признаков в узле

Чтобы достичь компромисса между точностью и производительностью возьмем размер леса равным 250, а долю признаков 0,01. В этом случае точность составит 54,05%. Это по-прежнему хуже результата нейронных сетей, но почти на 10% лучше, чем у дерева классификации. В отличие от нейронных сетей мы можем посмотреть на важность каждого слова. Так, среди первых 15 в основном прилагательные в негативном или позитивном свете описывающие заведение. Никаких названий блюд, музыки или других особенностей заведений, которые бы значимо влияли на результат не найдено.

Как мы выяснили, лучшие результаты получаются если проводить разбиение по 1% от всех признаков. В нашем случае это 50 наблюдений. Но так как исходная матрица является разреженной, то большая часть этих признаков может оказаться не информативна для некоторых наблюдений. В таких случаях принято применять методы снижения размерности. Мы будем использовать усеченное сингулярное разложение и метод главных компонент. Лучшие результаты приведены в таблице 3.

Таблица 3- Результаты случайных лесов

Параметры модели	Метод снижения размерноти	Точность на обучающем множестве	Точность на тестовом множестве
Случайный лес	SVD (100	99,96%	51, 93%

300 деревьев, 10 признаков в узле	компонент)		
Случайный лес 200 деревьев, 10 признаков в узле, максимальная глубина 15, минимальное количество наблюдений в узле 10, минимальная загрязненность в узле 0.01	SVD (100 компонент)	81,07%	51, 14%
Случайный лес 150 деревьев, 20 признаков в узле, максимальная глубина 50, минимальное количество наблюдений в узле 5, минимальная загрязненность в узле 0.01	SVD (100 компонент)	96,63%	52,02%
Экстремально случайные леса 150 деревьев, 20 признаков в узле, максимальная глубина 50, минимальное количество наблюдений в узле 5, минимальная загрязненность в узле 0.01	SVD (100 компонент)	98,35%	50,89%
Случайный лес 150 деревьев, 20 признаков в узле, максимальная глубина 50, минимальное количество наблюдений в узле 5, минимальная загрязненность в узле 0.01	PCA (100 компонент)	96,82%	52,25%

Первое, что бросается в глаза – сильное переобучение. Однако, попытки упростить модель сильно сказываются на точности. Далее сделаем случайный поиск по сетке для градиентного бустинга. Обучаться будем на выборке, к которой применили метод главных компонент. Параметры сетки:

- Количество деревьев: 50,100,150,200,300,500
- Скорость обучения: 0.01,0.03,0.05,0.1,0.15,0.2,0.3
- Максимальная глубина: 2,3,4,5,6,10
- Минимальное количество наблюдений в узле: 1,2,3,4,5,10
- Количество атрибутов в узле: 5,10,15,20

Точность лучшей модели на обучающем и тестовом множествах 60,99% и 55,21% соответственно. Как видим, градиентный бустинг заметно меньше переобучается и при этом дает лучшие результаты, чем случайные леса. В следующем подразделе заменим мешок слов на Word2Vec.

3.4 Word2Vec и сверточные нейронные сети

Мы уже использовали векторное представление слов, когда обучали рекурентные сети. Теперь создадим более сложную модель. Так же, как и раньше удалим лишнее из отзывов и сделаем лемматизацию. Будем включать в модель только те слова, которые встретились чаще, чем 30 раз, размер окна равен 10, а каждое слово будет представлено вектором размерности 300. Чтобы проверить, хорошее или нет представление мы получили, выберем несколько слов и будем искать для них ближайшие в векторном пространстве. Так, для слова пицца ближайшими оказались разновидности пиццы: кальцоне, маргарита, пепперони и другие, а для слова вкусный ближайшими оказались другие прилагательные, описывающие вкус блюда. Также считается, что если модель обучена хорошо, то слова можно складывать и вычитать. Прибавим к слову Италия слово круассан и вычтем слово пицца. Ближайшим словом оказалось слово Франция. Пример такой операции представлен на рисунке 20.

```
In [105]: model.most_similar_cosmul(positive=['italy', 'croissant'], negative=['pizza'])

Out[105]: [('france', 0.9873804450035095),
           ('chocolat', 0.9546630382537842),
           ('paris', 0.9524206519126892),
           ('croissants', 0.952370285987854),
           ('parisian', 0.9442649483680725),
           ('aux', 0.9347772002220154),
           ('tartine', 0.9069862365722656),
           ('raisin', 0.8985605835914612),
           ('jules', 0.8976535797119141),
           ('eclair', 0.8920069336891174)]
```

Рисунок 20 - Проверка Word2Vec

Если на вход сверточной нейронной сети подавать матрицу, где каждая строка является вектором соответствующего ей слова, то модели ошибались очень часто и дава-

ли точность чуть больше 20%. Поэтому преобразуем матрицу в вектор, каждая координата которого является средним значением всех элементов столбца. Лучшей из тестированных архитектур оказалась следующая: первый скрытый слой – слой свертки с 256 фильтрами, следующий слой – глобальный пулинг с выбором максимального значения, далее полносвязный слой со 128 нейронами и дропаутом, последний слой с 5 нейронами. Такая модель дала точность 58% на тестовой выборке, что всего на процент лучше результата логистической регрессии на тех же данных.

Несмотря на то, что Word2Vec не помог улучшить результаты в данной задаче, он сам по себе может являться довольно полезным инструментом, который помогает анализировать отрасль. Например, можно понять, что вызывает у потребителей больше положительных эмоций. Рассмотрим это на примере, пиццы и десертов. Используя алгоритм t-sne[21], отобразим необходимые слова в двумерном пространстве.

Рисунок 21 - Визуализация Word2Vec

Как видно из рисунка 21 слова, характеризующие вкус блюда с положителной стороны, расположены ближе к десертам. Маргарита же находится так далеко от

[21] Метод визуализации многомерных пространств, представленный в 2008 году Laurens van der Maaten в [41]

пиццы скорее всего по той причине, что является еще и напитком. Таким образом, можно отслеживать реакцию клиентов на новинки рынка.

Но как узнать, что искать? Рассмотрим два подхода. Первый – тематическое моделирование. Для этого, например, можно использовать латентное размещение Дирихле. Построим такую модель для положительных отзывов (4 и 5 звезд), которая ищет 15 тем. Темы представляют собой список слов с весами их важности. Пример такой темы показан на рисунке 22.

Эту тему можно охарактеризовать, как завтраки. Видно, что самые популярные блюда — это сэндвичи, яйца с беконом, оладьи, тосты и кофе. При этом отсутствуют каши, ягоды и фрукты. Можно предположить, что в Питтсбурге не очень популярно здоровое питание на завтрак. Другие темы, которые нашла модель: бары, в которых в основном продают алкоголь и показывают спортивные матчи, острые тайские и индийские блюда, мексиканская кухня, булочные, фастфуд, ночные клубы и другие.

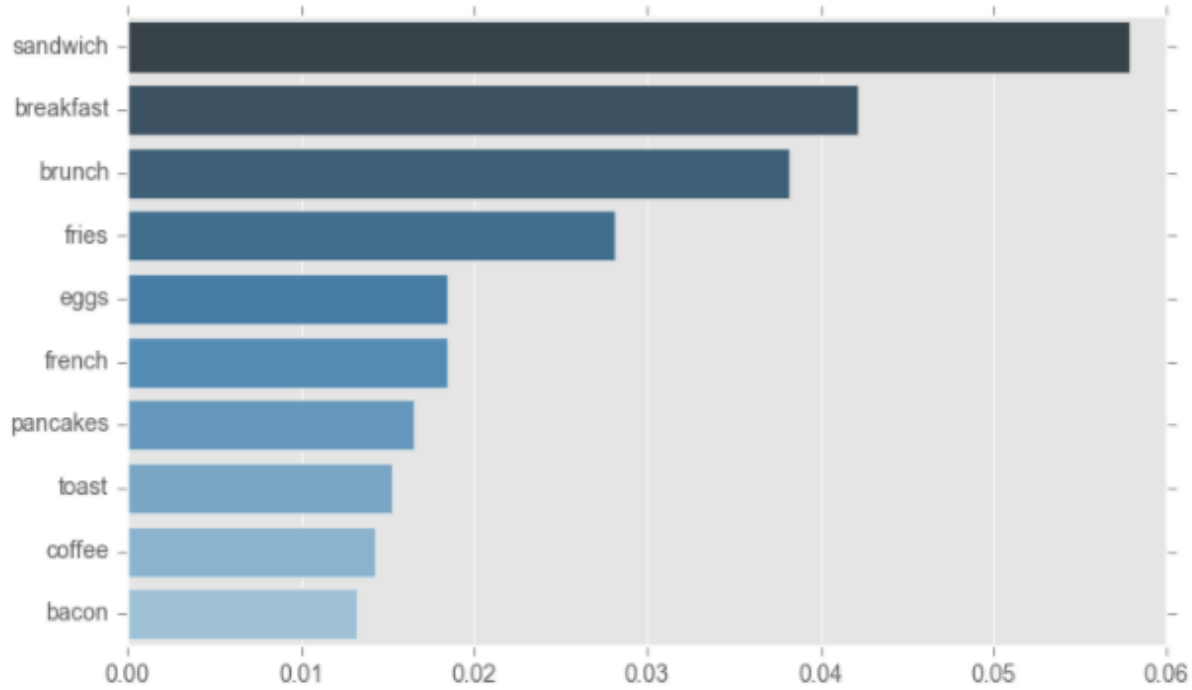


Рисунок 22 - Пример темы, полученной с помощью LDA

Второй способ – облака слов. Создадим их для отзывов с разными оценками отдельно. На рисунке 23 приведен пример такого облака для отзывов с оценкой 5. Как видно чаще хвалят еду, в то время как в негативных отзывах в основном ругают

обслуживание. Облака слов для других оценок представлены в приложении 3. Что касается ассортимента, то если в негативных отзывах чаще встречается китайская и тайская кухни, то в положительных чаще упоминают алкогольные напитки и более изысканные продукты: козий сыр, стейки, пирог из краба. Также в положительных комментариях можно встретить информацию о наличии дополнительных опций: вегетарианское меню, продукты без глютена.

Полученные модели чаще всего ошибались в соседних оценках. Возможно, эти оценки являются субъективными и разные люди оценили бы один и тот же отзыв по-разному. Проведем небольшой эксперимент и попросим людей предположить для 10 отзывов, каким оценкам они соответствуют. С отзывами из опроса можно ознакомиться в приложении 4. По результатам опроса средняя точность составила 47,69%, а медиана составила ровно половину правильных ответов. Далее на тех же вопрос была протестирована нейронная сеть, которая отгадала 8 из 10 оценок.

Рисунок 23 - облако слов для отзывов с оценкой 5

В результате этого небольшого эксперимента было решено объединить отзывы с оценками 1 и 2 в класс негативных отзывов, а 4 и 5 в класс позитивных отзывов. Сделаем мешок слов с TF-IDF размером 7000 слов и обучим на нем нейронную сеть прямого распространения. Точность на обучающей и тестовой выборках составила 85,62% и 83,28% соответственно.

3.5 Применение полученных моделей для формирования маркетинговой стратегии

На конкретном примере покажем применимость предлагаемых моделей для маркетингового исследования. Будем рассматривать город Монреаль. Первое, что стоит отметить – меньшее количество отзывов, даже несмотря на то, что в Монреале почти на 1000 заведений питания больше, нежели в Питтсбурге. Выберем позитивные и негативные отзывы с помощью нашей модели для заведения Schwartz's. Это старое и известное заведение, которое позиционируется как место, где можно попробовать классический канадский рецепт копченого мяса. Большинство его посетителей являются туристами. Поэтому для заведения важно поддерживать репутацию в интернете. Всего для этого ресторана на сайте Yelp есть 1290 отзывов, а средний рейтинг заведения 4 звезды. Посмотрим, что можно исправить, чтобы увеличить рейтинг до 5.

Точность работы модели для данного заведения составила 84%. Создадим облака слов для предсказанных и фактических негативных комментариев.

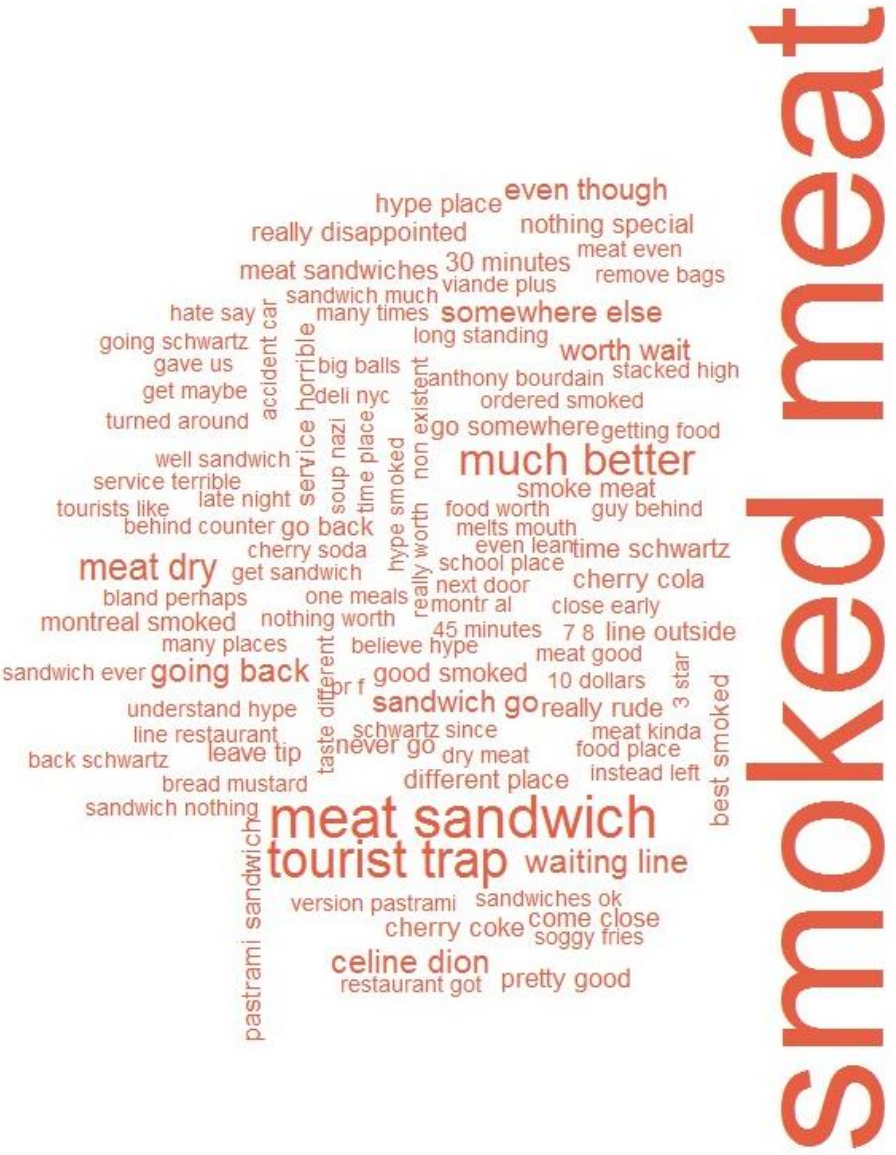


Рисунок 24 – Облако слов для негативных комментариев, предсказанных моделью

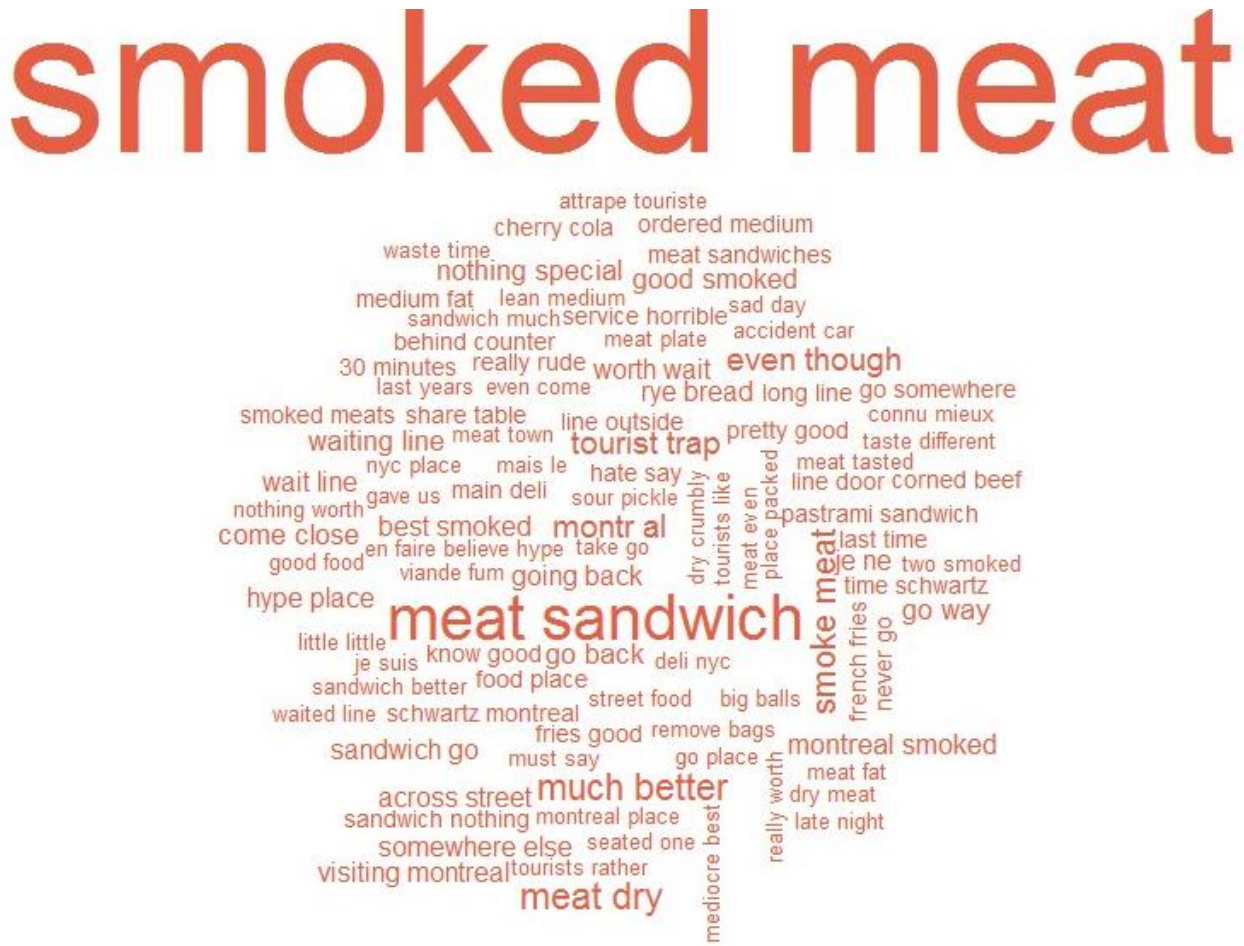


Рисунок 25 – Облако слов для фактических негативных комментариев

На рисунках 24 и 25 можно выделить одни и те же основные претензии посетителей:

- Плохое обслуживание
- Долгое время нахождения в очереди: 30-45 минут
- Сухое мясо
- Тесное помещение

Для того чтобы воспользоваться тематическим моделированием необходимо большее количество отзывов. Поэтому далее будем использовать Word2Vec, чтобы выяснить предпочтения посетителей. Обучим модель на всех отзывах в Монреале и начнем анализ.

В заведении есть основное блюдо, но нет никакого альтернативного выбора. Посмотрим, какие блюда можно предложить клиентам, которые жалуются на сухое мясо, так, чтобы при этом оставаться в рамках позиционирования заведения. Найдем ближайшие слова для суммы “копченый”,” мясо” и “лучший”. Ближайшим оказалось слово “пастрома”, которая уже используется в меню заведения. Попробуем теперь из

этой суммы вычесть вектор, соответствующий слову “сухой”. Ближайшими оказались векторы слов “осетр”,” лосось”,” квашеная капуста” и “лук-порей”. Так как “осетр” и “лосось” входили и в предыдущий список ближайших слов, то можно рекомендовать добавить сэндвичи с рыбой в меню, чтобы клиенты, опасающиеся получить сухое мясо выбирали их.

Основная проблема заведения, конечно, заключатся в плохом сервисе. Если прочитать несколько негативных комментариев, то становится понятно, что плохое обслуживание связано с тем, что официантам необходимо спешить обслужить очень большое количество людей. Если посмотреть слова, которые чаще всего встречаются со словосочетанием “хороший сервис”, то ближайшими окажутся “персонализированный”, “учтивый” и “быстрый”, и только через несколько десятков слов такие слова как “дружелюбный”, “юмор” и “позитивный”. Таким образом, компании стоит рассмотреть вопрос о формировании культуры общения с клиентами, а также оптимизировать процесс обслуживания посетителей. Например, можно ввести оплату не только наличными, но и с помощью карт и смартфонов, проверить насколько удобно расположение всех продуктов и приборов на кухне, нанять помощников кассиров в зал, увеличить количество касс или же дать возможность сделать заказ непосредственно через свой сайт или специальный автомат. Но даже все эти меры не помогут полностью избавиться от очередей. Поэтому необходимо сделать так, чтобы ожидание в этом кафе не доставляло дискомфорта. В частности, так как посетители жаловались на тесное помещение, то стоит рассмотреть варианты с изменением планировки заведения, увеличения площади за счет размещения части клиентов на улице. Найдем слова, которые ассоциируются у посетителей со словом бар: “камин”, “приглушенный свет”, “вместительный”, ”лофт”,”стильный”. Конечно, вряд ли первым делом владельцам заведения стоит покупать камин, но сделать качественный ремонт, в соответствии с найденными признаками было бы неплохо. Наконец, посмотрим, какими словами люди описывают заведения, которым ставят 5 звезд. Среди первых 20 ближайших векторов к сумме “5” и “звезды” есть лишь одно прилагательное, которое описывает какое-то конкретное качество заведения – “чистота”. Если посмотреть фотографии, которыми делятся пользователи, то можно подтвердить вывод о необходимости ремонта и новых правилах поддержания порядка в этом заведении.

Для данного ресторана доля неудовлетворенных потребителей, оценивших заведение в 1,2 и 3 звезды составляет около 23%. Таким образом, почти каждый чет-

вертый остался недоволен посещением, что, конечно, негативно сказывается на репутации Schwartz's. С другой стороны, это означает, что существует потенциал для роста. В работе [44] было установлено, что увеличение рейтинга ресторана на одну звезду приводит к увеличению дохода на 5-9% в зависимости от особенности ресторанов. Так как Schwart’z не обязан публиковать свое отчетность, то найти актуальную информацию о доходах не представляется возможным. Но в 2014 году доход составил 9,2 миллиона канадских долларов. [45] Таким образом, в случае успешных изменений можно ожидать увеличение годового дохода на 460-920 C$. Конечно, представленная методология не претендует на то, чтобы полностью заменить опросы потребителей, но она позволяет значительно ускорить процесс анализа мнений за счет автоматического определения тональности отзывов, наглядной визуализации ключевых проблем и достоинств организации и всего рынка, а также выявления предпочтений с помощью Word2Vec. Такой подход помогает сформировать ряд гипотез, которые могут быть проверены в процессе непосредственного опроса, либо, минуя этот этап, измерением ключевых метрик после внедрения изменений. Также стоит отметить, что предложенные методы применимы не только в ресторанном бизнесе, но и большинстве других сфер.

Заключение

В ходе данной работы были рассмотрены основные сферы применения методов обработки текстовых данных в бизнесе, а также найдены конкретные примеры их применения в маркетинге. Были опробованы различные методы обработки текстов и определения их тональности. Лучший результат показали нейронные сети прямого распространения, на вход которым подавался мешок слов с биграммами с TF-IDF взвешиванием и размером словаря 10 000 слов, а также рекуррентные нейронные сети с GRU нейронами. Также было установлено, что оценка по пятибалльной шкале довольно субъективна и каждый человек может оценить один и тот же текст по-разному. В проведенном эксперименте нейронная сеть сработала даже несколько точнее, чем участники опроса.

Результаты проведенного исследования позволяют сделать вывод, что с помощью машинного обучения возможно достичь приемлемой точности для задачи распознавания тональности высказывания, что позволяет в реальном времени отслеживать реакцию пользователей на продукты и деятельность компании. Данные для обучения моделей можно получить с помощью API крупных сервисов рекомендаций. Для заведений в России в частности может быть использован Foursquare API или TripAdvisor API. В дальнейшем для более широкой картины можно брать комментарии и посты пользователей в социальных сетях. Чтобы получить представление о том, что именно нравится пользователям в какой-то сфере, а что вызывает у них раздражение можно воспользоваться тематическим моделированием, облаками слов и Word2Vec. Предложенные инструменты позволяют частично автоматизировать данный процесс и узнать мнение большего числа клиентов, нежели обычные опросы. Но учитывая тренды поведения пользователей социальных сетей, для будущих исследований интерес представляет анализ тональности фотографий и видеозаписей пользователей, а также коротких текстовых сообщений, которые относятся к этим фотографиям и видео. Так по результатам исследований в 2017 году доля видео контента от всей информации в сети составит 74%, твитами с картинками делятся в 1,5 раза чаще, чем без картинок, в Facebook посты с картинками собирают просмотров больше в 2.3 раза, в декабре 2016 года Instagram, платформа, содержащая больше всего фотографий и коротких видео, объявила о росте пользовательской базы на 100 миллионов за полгода, а на данный момент сервисом пользуется более 700 миллионов человек.[42][43] Таким образом, гораздо больше информации можно получить, анализируя не только тексты, но и фото и видео контент.

Список использованных источников

1) Top H-Index for Computer Science and Electronics URL: http://www.guide2research.com/scientists/ дата обращения:08.03.2017
2) Moore, Gordon E. (1965). "Cramming more components onto integrated circuits"
3) Top500 List - June 1997 URL: https://www.top500.org/list/1997/06/300/?page=3 дата обращения:12.03.2017
4) Yelp Case Study: https://aws.amazon.com/ru/solutions/case-studies/yelp/ дата обращения:12.03.2017
5) CS224d: Deep Learning for Natural Language Processing URL: http://cs224d.stanford.edu/ дата обращения:12.03.2017
6) Deep Learning for Natural Language Processing URL: 2016-2017 URL: https://www.cs.ox.ac.uk/teaching/courses/2016-2017/dl/ дата обращения:12.03.2017
7) MIT OPEN COURSE WARE URL: 2016-2017 URL: https://ocw.mit.edu/courses/find-by-department/ дата обращения:12.03.2017
8) Michele Chambers, Christine Doig, Ian Stokes-Rees. Breaking Data Science Open. How Open Data Science Is Eating the World. 2017
9) CRISP-DM 1.0 Step-by-step data mining guide Pete Chapman (NCR), Julian Clinton (SPSS), Randy Kerber (NCR), Thomas Khabaza (SPSS), Thomas Reinartz (DaimlerChrysler), Colin Shearer (SPSS) and Rüdiger Wirth (DaimlerChrysler)
10) AMA: Yann LeCun URL: https://www.reddit.com/r/MachineLearning/comments/25lnbt/ama_yann_lecun/ дата обращения 28.03.2017
11) Boosting Sales With Machine Learning. Xeneta Blog. URL: https://medium.com/xeneta/boosting-sales-with-machine-learning-fbcf2e618be3#.cr60ssq8w дата обращения 26.03.2017
12) Natural Language Processing Market by Type (Rule-Based, Statistical, and Hybrid), Technologies (Recognition, IVR, OCR, Speech Recognition, Text Processing, Pattern & Image Recognition), by Deployment Type, Vertical & by Region - Global Forecast to 2021
13) HSE: Customer Satisfaction Index https://www.hse.ru/data/2013/12/06/1336473230/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D0%BA%D0%B0%20CSI.pdf дата обращения 29.03.2017
14) Stewart Butterfield interview URL: http://firstround.com/review/From-0-to-1B-Slacks-Founder-Shares-Their-Epic-Launch-Strategy/ дата обращения 26.03.2017
15) URL: https://techcrunch.com/2016/04/01/rocketship-emoji/ дата обращения 26.03.2017
16) Fornell, C., S. Mithas, F.V. Morgeson III, and M.S. Krishnan (2006). "Customer Satisfaction and Stock Prices: High Returns, Low Risk," Journal of Marketing, 70(1), 3−14.
17) The use of sentiment analysis tools in online reputation management process. Borcan, A. Eindhoven University of Technology, 2013
18) Text Analytics as Commodity: обзор приложений текстовой аналитики URL: https://habrahabr.ru/company/textocat/blog/259035/ Дата обращения 3.04.2017
19) Documentation is for scikit-learn. 4.2 Feature extraction. URL: http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction Дата обращения 9.04.2017
20) TF-IDF URL: https://ru.wikipedia.org/wiki/TF-IDF Дата обращения 09.04.2017

21) Нугуманова А.Б., Бессмертный И.А., Байбурин Е.М., Пецина П. (2016). Обогащение модели Bag of words семантическими связями для повышения качества классификации текстов предметной области.
22) Специализация “ Машинное обучение и анализ данных”. Математика и Python для анализа данных. Матричные разложения. URL: https://www.coursera.org/learn/mathematics-and-python/lecture/zuTEu/sviaz-singhuliarnogho-razlozhieniia-i-priblizhieniia-matritsiei-mien-shiegho Дата обращения: 9.04.2017
23) CS 224D: Deep Learning for NLP. Lecture Notes: Part I. Spring 2016. Francois Chaubard, Rohit, Mundra, Richard Socher.
24) Специализация “ Машинное обучение и анализ данных”. "Поиск структуры в данных. Понижение размерности и матричные разложения. URL: https://www.coursera.org/learn/unsupervised-learning/lecture/e72bH/mietod-ghlavnykh-komponient-rieshieniie Дата обращения 10.04.2017
25) Отношение Рэлея URL: https://ru.wikipedia.org/wiki/Отношение_Рэлея Дата обращения 10.04.2017
26) Google Open Source Blog URL: https://opensource.googleblog.com/2013/08/learning-meaning-behind-words.html Дата обращения: 10.04.2017
27) Facebook Research Github URL: https://github.com/facebookresearch/fastText Дата обращения: 10.04.2017
28) T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean. Distributed Representations of Words and Phrases and their Compositionality. 2013
29) A. Joulin, E. Grave, P. Bojanowski, T. Mikolov. 2016. Bag of Tricks for Efficient Text Classification
30) Faceook Research Github URL: https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md дата обращениия 10.04.2017
31) Word2Vec Tutorial - The Skip-Gram Model. URL: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/ дата обращения 11.04.2017
32) Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov. FastText. URL: https://research.fb.com/fasttext/ Дата обращения: 11.04.2017
33) Scikit-learn documentation. 1.10. Decision Tree URL: http://scikit-learn.org/stable/modules/tree.html дата обращения 11.04.2017
34) Condorcet's jury theorem URL: https://en.wikipedia.org/wiki/Condorcet%27s_jury_theorem Дата обращения 12.04.2017.
35) P. Geurts, D. Ernst, L. Wehenkel. Extremely randomized trees. 2006.
36) N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. 2013.
37) BaseGroup Labs. Алгоритм обратного распространения ошибки URL: http://www.stgau.ru/company/personal/user/7684/files/lib/BackProp.pdf дата обращения 16.04.17
38) Методы оптимизации нейронных сетей URL: https://habrahabr.ru/post/318970/ Дата обращения 17.04.17
39) Семинары по выбору моделей. Выбор моделей и критерии качества. URL: http://www.machinelearning.ru/wiki/images/1/1c/Sem06_metrics.pdf Дата обращения: 17.04.17

40) Understanding LSTM Networks. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ дата обращения 18.04.2017
41) Laurens van der Maaten. Visualizing Data using t-SNE. 2008
42) Visual Content Marketing Statistics You Should Know in 2017 URL: https://blog.hubspot.com/marketing/visual-content-marketing-strategy#sm.0001s96l3kuv4emwukg1i1o7lqdyh Дата обращения: 02.05.2017
43) Instagram Blog. URL: http://blog.instagram.com/post/154506585127/161215-600million Дата обращения 02.05.2017
44) Michael Luca. 2016. Reviews, Reputation, and Revenue: The Case of Yelp.com. Working Paper 12-016
45) Schwartz's . URL: https://en.wikipedia.org/wiki/Schwartz%27s Дата обращения 08.05.2017

Приложение 1. Пример используемых данных.

```
df_rev.head(3)
```

	review_id	text	votes.cool	business_id	votes.funny	stars	date	type	votes.useful
xDAcGqfsorJp3Q	Ya85v4eqdd6k9Od8HbQjyA	Mr Hoagie is an institution. Walking in, it do...	0	5UmKMjUEUNdYWqANhGckJw	0	4	2012-08-01	review	0
iR4A0wspR9BYHA	KPvLNJ21_4wbYNctrOwWdQ	Excellent food. Superb customer service. I mis...	0	5UmKMjUEUNdYWqANhGckJw	0	5	2014-02-13	review	0
W42h6alXgFxAXQ	fFSoGV46Yxuwbr3fHNuZig	Yes this place is a little out dated and not o...	0	5UmKMjUEUNdYWqANhGckJw	1	5	2015-10-31	review	1

Рисунок 1П – Пример таблицы с отзывами

```
df_bus.head()
```

	attributes.Ambience.divey	attributes.Dietary Restrictions.vegan	attributes.Happy Hour	hours.Thursday.open	attributes.Order at Counter	attributes.Hair Types Specialized In.africanamerican	attributes.Hair Types Specialized In.kids	attribu
0	False	NaN	NaN	11:00	NaN	NaN	NaN	NaN
1	NaN	NaN	True	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	True	NaN	False	10:00	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	11:00	NaN	NaN	NaN	NaN

5 rows × 98 columns

Рисунок 2П – Пример таблицы с информацией о компании

```
"Went for breakfast on 6/16/14. We received very good service and meal came within a few minutes.Waitress could have smiled mor
e but was friendly. \nI had a Grand Slam... it was more than enough food. \nMeal was very tasty... We will definitely go back.
 \nIt is a popular Denny's."
```

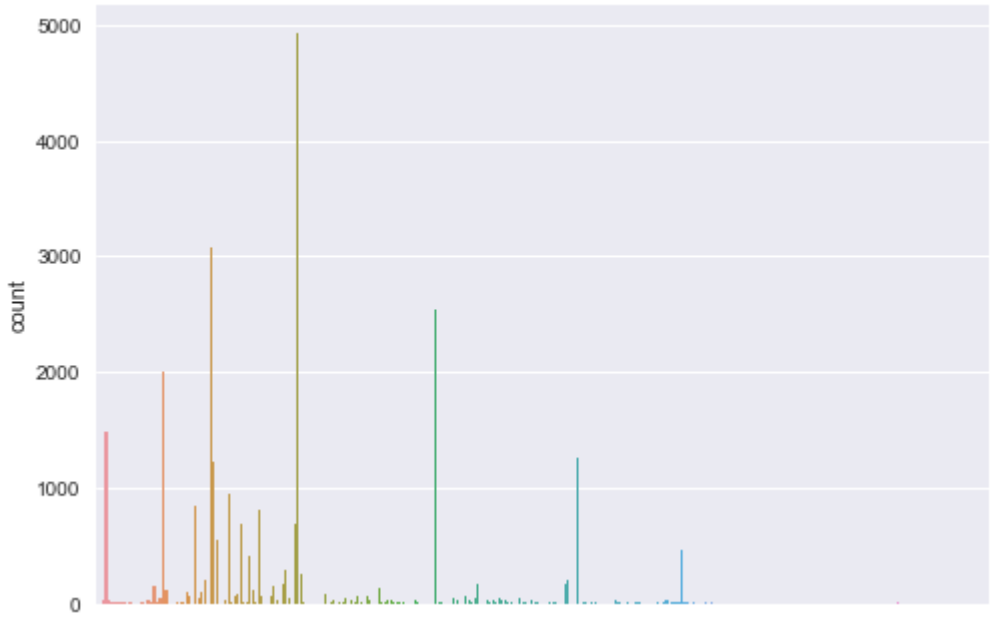
Рисунок 3П – Пример отзыва

Приложение 2. Реализация в Python

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
%matplotlib inline
matplotlib.style.use('ggplot')
```

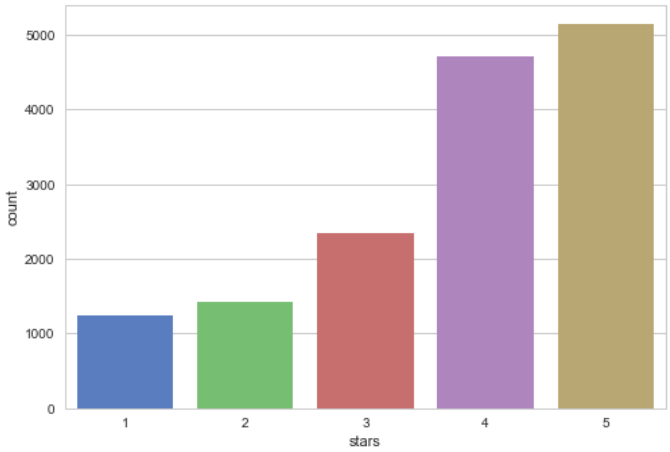
```
import seaborn as sns

df_rev=pd.read_csv("D:\\Yelp\\yelp_academic_dataset_review.csv",sep=",",decimal=".")
df_bus=pd.read_csv("D:\\Yelp\\yelp_academic_dataset_business.csv",sep=",",decimal=".
")
num=list(map(lambda x : "Restaurants" in x,df_bus.loc[:,"categories"]))
business_id_restaraunt=list(df_bus.loc[num,"business_id"])
ax = sns.countplot(df_bus[num].city)
```




```
df_bus[num].city.value_counts()[0:10]
Las Vegas             4928
Phoenix               3082
Montréal              2546
Charlotte             2003
Pittsburgh            1489
Edinburgh             1266
Scottsdale            1231
Mesa                   944
Madison                844
Tempe                  816

num2=list(map(lambda x,y : x=="Pittsburgh" and "Restaurants" in
y,df_bus.city,df_bus.categories))
business_id_restaraunt=list(df_bus.loc[num2,"business_id"])
indexRestInRev=df_rev.business_id.isin(business_id_restaraunt)
dfRevRest=df_rev[:][indexRestInRev]
textAndStars=dfRevRest[['stars','text']]
y=textAndStars.stars
x=textAndStars.text
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=1234)
sns.set(style="whitegrid",palette="muted")
ax = sns.countplot(y_test)
```




```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer(stop_words='english',max_features=5000)
X_train_counts = count_vect.fit_transform(X_train)
X_test_count=count_vect.transform(X_test)
from sklearn.feature_extraction.text import TfidfTransformer
tfidf = TfidfTransformer()
X_train_tfidf = tfidf.fit_transform(X_train_counts)
X_test_tfidf = tfidf.fit_transform(X_test_count)
X_train_tfidf_notSparse=X_train_tfidf.toarray()
y_test_val=y_test.values
y_train_val=y_train.values

y_train_cat=np_utils.to_categorical(y_train_val,5)
y_test_cat=np_utils.to_categorical(y_test_val,5)

model=Sequential()
model.add(Dense(800,input_dim=5000,init="normal",activation="relu"))
model.add(Dense(5,init="normal",activation="softmax"))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
print(model.summary())
model.fit(X_train_arr_tfidf,y_train_cat,batch_size=200,nb_epoch=3,
      verbose=1)
Epoch 1/3
59436/59436 [==============================] - 54s - loss: 1.0511 - acc: 0.5401

Epoch 2/3
59436/59436 [==============================] - 61s - loss: 0.8259 - acc: 0.6409

Epoch 3/3
59436/59436 [==============================] - 59s - loss: 0.7410 - acc: 0.6877

score=model.evaluate(X_test_tfidf.toarray(),y_test_cat,verbose=0)
score[1]*100
Out[15]: 58.133117976841064
model3=Sequential()
model3.add(Dense(800,input_dim=5000,init="normal",activation="relu"))
model3.add(Dense(5,init="normal",activation="softmax"))
model3.compile(loss='categorical_crossentropy',optimizer='Nadam',metrics=['accuracy'])
```

```
print(model.summary())
model3.fit(X_train_arr_tfidf,y_train_cat,batch_size=200,nb_epoch=3,verbose=1)
```

```
Epoch 3/3
59436/59436 [==============================] - 59s - loss: 0.6444 - acc: 0.7419
```

```
score=model3.evaluate(X_test_tfidf.toarray(),y_test_cat,verbose=0)
score[1]*100
```

```
Out[22]: 57.042869642498239
```

```
model4=Sequential()
model4.fit(X_train_arr_tfidf,y_train_cat,batch_size=200,nb_epoch=3,verbose=1)
```

```
Epoch 3/3
59436/59436 [==============================] - 22s - loss: 0.1359 - acc: 0.3296
```

```
model5.fit(X_train_arr_tfidf,y_train_cat,batch_size=200,nb_epoch=3,verbose=1)
```

```
Epoch 3/3
59436/59436 [==============================] - 35s - loss: 0.7805 - acc: 0.6687
```

```
score=model5.evaluate(X_test_tfidf.toarray(),y_test_cat,verbose=0)
score[1]*100
```

```
Out[13]: 58.153307760810378
```

```
model6.fit(X_train_arr_tfidf,y_train_cat,batch_size=200,nb_epoch=3,verbose=1)
```

```
Epoch 3/3
59436/59436 [==============================] - 45s - loss: 0.9158 - acc: 0.5120
```

```
score=model6.evaluate(X_test_tfidf.toarray(),y_test_cat,verbose=0)
score[1]*100
```

```
Out[22]: 52.742445657034885
```

```
model9.fit(X_train_arr_tfidf,y_train_cat,batch_size=500,nb_epoch=3,verbose=1)
```

```
Epoch 3/3
59436/59436 [==============================] - 25s - loss: 0.6023 - acc: 0.7596
```

```
score=model9.evaluate(X_test_tfidf.toarray(),y_test_cat,verbose=0)
score[1]*100
```

```
Out[33]: 56.242008211514985
```

```
X_train.text[0]
```

```
"Went for breakfast on 6/16/14. We received very good service and meal came wit
hin a few minutes.Waitress could have smiled more but was friendly. \r\r\nI had
a Grand Slam... it was more than enough food. \r\r\nMeal was very tasty... We w
ill definitely go back. \r\r\nIt is a popular Denny's."
```

```
import re
from nltk.corpus import stopwords
def review_to_words( raw_review ):
    # Убираем все кроме букв и восклицательного знака
    letteronly = re.sub("[^a-zA-Z!]", " ", raw_review)
    letteronly = re.sub("!"," !",letteronly)
    # Переводим все слова в нижний регистр
    words = letteronly.lower().split()
    # создаем список стопслов
    stops = set(list([x for x in stopwords.words("english") if not x in
['no','not','nor','only','very','too']]))
    #удаляем стопслова
    mwords = [w for w in words if not w in stops]
    # объединяем все слова обратно в предложение
```

```
    return( " ".join(mwords ))

clean_review = review_to_words(X_train.text[0] )
print (clean_review)
 went breakfast received very good service meal came within minutes waitress cou
 ld smiled friendly grand slam enough food meal very tasty definitely go back po
 pular denny
clean_X_train= []
clean_X_test= []
model.
fit(X_train_counts_clean.toarray(),y_train_cat,batch_size=200,nb_epoch=2,verbose=1)
 Epoch 2/2
 59436/59436 [==============================] - 40s - loss: 0.7009 - acc: 0.7123

score=model.evaluate(X_test_counts_clean.toarray(),y_test_cat,verbose=0)
score[1]*100
 Out[75]: 58.994548759331124
model2=Sequential()
model2.add(Dense(800,input_dim=5000,init="normal",activation="relu"))
model2.add(Dense(6,init="normal",activation="softmax"))
model2.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
mod-
el2.fit(X_train_counts_clean.toarray(),y_train_cat,batch_size=500,nb_epoch=3,verbose=1)
 Epoch 2/3
 59436/59436 [==============================] - 23s - loss: 0.7535 - acc: 0.6923
score=model2.evaluate(X_test_counts_clean.toarray(),y_test_cat,verbose=0)
score[1]*100
 58.711891783961335
X_train_tfidf_clean = tfidf.fit_transform(X_train_counts_clean)
X_test_tfidf_clean = tfidf.transform(X_test_counts_clean)

model10=Sequential()
model10.add(Dense(800,input_dim=5000,init="normal",activation="relu"))
model10.add(Dense(6,init="normal",activation="softmax"))
model10.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
print(model10.summary())
model10.fit(X_train_tfidf_clean.toarray(),y_train_cat,batch_size=500,nb_epoch=3,
      verbose=1)
 Epoch 3/3
 59436/59436 [==============================] - 23s - loss: 0.7655 - acc: 0.6761

score=model10.evaluate(X_test_tfidf_clean.toarray(),y_test_cat,verbose=0)
score[1]*100
 59.169526887265725
model11.fit(X_train_tfidf_clean.toarray(),y_train_cat,batch_size=30,nb_epoch=1,
      verbose=1)
 Epoch 1/1
 59436/59436 [==============================] - 228s - loss: 0.9525 - acc: 0.575
 7
score=model11.evaluate(X_test_tfidf_clean.toarray(),y_test_cat,verbose=0)
score[1]*100
```

```
59.815599974083121
```

```
vectorizer = CountVectorizer(analyzer = "word", tokenizer = None, ngram_range=(1,2), preprocessor = None, stop_words = None, max_features = 5000)
X_train_counts_clean_bigram = vectorizer.fit_transform(clean_X_train)
X_test_counts_clean_bigram = vectorizer.transform(clean_X_test)
model2bi.fit(X_train_counts_clean_bigram.toarray(), y_train_cat, batch_size =512, nb_epoch=2, verbose=1)
```

```
Epoch 2/2
59436/59436 [==============================] - 14s - loss: 0.7794 - acc: 0.6754
```

```
score=model2bi.evaluate(X_test_counts_clean_bigram.toarray(),y_test_cat,verbose=0)
score[1]*100
```

```
Out[87]: 60.576081837127759
```

```
model2bi.fit(X_train_counts_clean_bigram.toarray(),  y_train_cat, batch_size=512, nb_epoch=2, verbose=1)
```

```
Epoch 2/2
59436/59436 [==============================] - 20s - loss: 0.9012 - acc: 0.6130
```

```
score=model2bi.evaluate(X_test_counts_clean_bigram.toarray(),y_test_cat,verbose=0)
score[1]*100
```

```
Out[96]: 61.235614780125267
```

```
model3bi=Sequential()
model3bi.add(Dense(600,input_dim=5000,init="normal",activation="relu"))
model3bi.add(Dropout(0.7))
model3bi.add(Dense(32,init="normal",activation="tanh"))
model3bi.add(Dropout(0.7))
model3bi.add(Dense(5,init="normal",activation="softmax"))
model3bi.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
model3bi.fit(X_train_counts_clean_bigram.toarray(),y_train_cat,batch_size=512, nb_epoch=4, verbose=1)
```

```
Epoch 4/4
59436/59436 [==============================] - 6s - loss: 0.8838 - acc: 0.6233
```

```
score=model3bi.evaluate(X_test_counts_clean_bigram.toarray(),y_test_cat,verbose=0)
score[1]*100
```

```
Out[36]: 61.390403123889989
```

```
model4bi.fit(X_train_tfidf_clean_bigram.toarray(),y_train_cat,batch_size=512, nb_epoch=2, verbose=1)
```

```
Epoch 2/2
59436/59436 [==============================] - 24s - loss: 0.8919 - acc: 0.6127
```

```
score=model4bi.evaluate(X_test_tfidf_clean_bigram.toarray(),y_test_cat,verbose=0)
score[1]*100
```

```
Out[125]: 61.85476815558529
```

```
from keras.layers import Merge
left_branch = Sequential()
left_branch.add(Dense(256, input_dim=5000,init="normal",activation="relu"))
left_branch.add(Dropout(0.5))
right_branch = Sequential()
right_branch.add(Dense(256, input_dim=5000,init="normal",activation="tanh"))
right_branch.add(Dropout(0.5))
```

```
merged = Merge([left_branch, right_branch], mode='concat')
final_model = Sequential()
final_model.add(merged)
final_model.add(Dense(5,init="normal", activation='softmax'))
final_model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
final_model.fit([X_train_tfidf_clean_bigram.toarray(), X_train_tfidf_clean_bigram.toarray()], y_train_cat,batch_size=512,nb_epoch=2, verbose=1)
```

```
Epoch 2/2
59436/59436 [==============================] - 9s - loss: 0.8483 - acc: 0.6327
```

```
score=final_model.evaluate([X_test_tfidf_clean_bigram.toarray(), X_test_tfidf_clean_bigram.toarray()], y_test_cat,verbose=0)
score[1]*100
```

```
61.814388587646675
```

```
vectorizer2 = CountVectorizer(analyzer = "word", tokenizer = None, ngram_range=(1,2),
preprocessor = None, stop_words = None,   max_features = 10000)
X_train_counts_clean_bigram2 = vectorizer2.fit_transform(clean_X_train)
X_test_counts_clean_bigram2 = vectorizer2.transform(clean_X_test)
X_train_tfidf_clean_bigram2 = tfidf.fit_transform(X_train_counts_clean_bigram2)
X_test_tfidf_clean_bigram2 = tfidf.transform(X_test_counts_clean_bigram2)
model4bi=Sequential()
model4bi.add(Dense(600,input_dim=10000,init="normal",activation="relu"))
model4bi.add(Dropout(0.7))
model4bi.add(Dense(6,init="normal",activation="softmax"))
model4bi.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
model4bi.fit(X_train_tfidf_clean_bigram2.toarray(),y_train_cat,batch_size=512, nb_epoch=3, verbose=1)
```

```
Epoch 3/3
59436/59436 [==============================] - 40s - loss: 0.7868 - acc: 0.6645
```

```
score=model4bi.evaluate(X_test_tfidf_clean_bigram2.toarray(),y_test_cat,verbose=0)
score[1]*100
```

```
Out[165]: 62.473921528237362
```

```
import keras.preprocessing.text
tk = keras.preprocessing.text.Tokenizer(nb_words=5000,lower=True, split=" ")
tk.fit_on_texts(clean_X_train)
X_train_from_dict=tk.texts_to_sequences(clean_X_train)
X_test_from_dict=tk.texts_to_sequences(clean_X_test)
X_train_from_dict = sequence.pad_sequences(X_train_from_dict, maxlen=460)
X_test_from_dict = sequence.pad_sequences(X_test_from_dict, maxlen=460)
model = Sequential()
model.add(Embedding(10000,32, input_length=490))
model.add(LSTM(100))
model.add(Dense(5, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
model.fit(X_train_from_dict,y_train_cat,batch_size=64,nb_epoch=3,verbose=1)
```

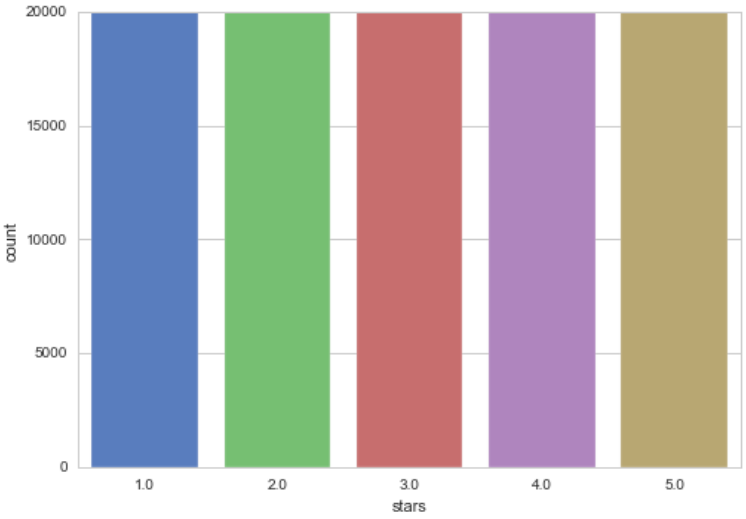
```
Epoch 3/3
59436/59436 [==============================] - 851s - loss: 0.8103 - acc: 0.647
```

```
score=model.evaluate(X_test_from_dict,y_test_cat,verbose=0)
score[1]*100
```

```
Out[98]:  59.936738676093896
model2 = Sequential()
model2.add(Embedding(5000,32, input_length=460))
model2.add(LSTM(100,dropout_W=0.2,dropout_U=0.2))
model2.add(Dense(5, activation='softmax'))
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model2.summary())
model2.fit(X_train_from_dict,y_train_cat,batch_size=32,nb_epoch=3,verbose=1)
Epoch 3/3
59436/59436 [==============================] - 1607s - loss: 0.8892 - acc: 0.60
score=model2.evaluate(X_test_from_dict,y_test_cat,verbose=0)
score[1]*100
Out[116]:  59.12914731972824

tk = keras.preprocessing.text.Tokenizer(nb_words=5000,lower=True, split=" ")
tk.fit_on_texts(clean_X_train)
X_train_from_dict=tk.texts_to_sequences(clean_X_train)
X_test_from_dict=tk.texts_to_sequences(clean_X_test)
X_train_from_dict = sequence.pad_sequences(X_train_from_dict, maxlen=448)
X_test_from_dict = sequence.pad_sequences(X_test_from_dict, maxlen=448)
model22 = Sequential()
model22.add(Embedding(5000,32, input_length=448))
model22.add(LSTM(100,dropout_W=0.2,dropout_U=0.2))
model22.add(Dense(5, activation='softmax'))
model22.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']
model22.fit(X_train_from_dict,y_train_cat,batch_size=32,nb_epoch=3,verbose=1)
Epoch 3/3
59436/59436 [==============================] - 1172s - loss: 0.8695 - acc: 0.61
53
score=model22.evaluate(X_test_from_dict,y_test_cat,verbose=0)
score[1]*100
Out[165]:  59.51948314313492
prediction=model22.predict_classes(X_test_from_dict)
cnf_matrix=confusion_matrix(y_test,prediction)
np.set_printoptions(precision=1)
plt.figure(figsize=(20,10))
plot_confusion_matrix(cnf_matrix, classes=['1 star','2 stars','3 stars','4 stars','5
stars'],normalize=True,
                      title='Confusion matrix, with normalization')
model3 = Sequential()
model3.add(Embedding(5000,32, input_length=448))
model3.add(LSTM(100,dropout_W=0.2,dropout_U=0.2,return_sequences=True))
model3.add(LSTM(32,dropout_W=0.2,dropout_U=0.2))
model3.add(Dense(5, activation='softmax'))
model3.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model3.summary())
model3.fit(X_train_from_dict,y_train_cat,batch_size=32,nb_epoch=3,verbose=1)
59436/59436 [==============================] - 1894s - loss: 0.8940 - acc: 0.60
23
score=model3.evaluate(X_test_from_dict,y_test_cat,verbose=0)
```

```
score[1]*100
Out[48]: 59.33777508500431
from keras.layers import GRU
modelGRU = Sequential()
modelGRU.add(Embedding(5000,32, input_length=460))
modelGRU.add(GRU(100,dropout_W=0.2,dropout_U=0.2))
modelGRU.add(Dense(5, activation='softmax'))
modelGRU.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
modelGRU.fit(X_train_from_dict,y_train_cat,batch_size=64,nb_epoch=3,verbose=1)
59436/59436 [==============================] - 973s - loss: 0.8502 - acc: 0.6209
score=modelGRU.evaluate(X_test_from_dict,y_test_cat,verbose=0)
score[1]*100
Out[66]: 61.006797228473076
ax = sns.countplot(y)
```




```
vectorizer = CountVectorizer(analyzer = "word", tokenizer = None, ngram_range=(1,2), preprocessor = None, stop_words = None, max_features = 2500)
X_train_counts_clean_bigram = vectorizer.fit_transform(clean_X_train)
X_test_counts_clean_bigram = vectorizer.transform(clean_X_test)
tfidf = TfidfTransformer()
X_train_tfidf_clean_bigram = tfidf.fit_transform(X_train_counts_clean_bigram)
X_test_tfidf_clean_bigram = tfidf.transform(X_test_counts_clean_bigram)
model2.
fit(X_train_tfidf_clean_bigram.toarray(),y_train_cat,batch_size=32,nb_epoch=2,verbose=1)
Epoch 2/2
90000/90000 [==============================] - 150s - loss: 0.8829 - acc: 0.615
score=model2.evaluate(X_test_tfidf_clean_bigram.toarray(),y_test_cat,verbose=0)
score[1]*100
Out[184]: 60.440000000000005
vectorizer = CountVectorizer(analyzer = "word", tokenizer = None,
ngram_range=(1,2),preprocessor = None, stop_words = None,  max_features = 10000)
X_train_counts_clean_bigram = vectorizer.fit_transform(clean_X_train)
X_test_counts_clean_bigram = vectorizer.transform(clean_X_test)
tfidf = TfidfTransformer()
X_train_tfidf_clean_bigram = tfidf.fit_transform(X_train_counts_clean_bigram)
X_test_tfidf_clean_bigram = tfidf.transform(X_test_counts_clean_bigram)
```

```
model3.
fit(X_train_tfidf_clean_bigram.toarray(),y_train_cat,batch_size=32,nb_epoch=2,verbose=1)
 Epoch 2/2
 90000/90000 [==============================] - 576s - loss: 0.7893 - acc: 0.665
score=model3.evaluate(X_test_tfidf_clean_bigram.toarray(),y_test_cat,verbose=0)
score[1]*100
 Out[189]: 62.029999999999994
vectorizer = CountVectorizer(analyzer = "word", tokenizer = None,  ngram_range=(1,2),
preprocessor = None, stop_words = None,  max_features = 15000)
model3.
fit(X_train_tfidf_clean_bigram.toarray(),y_train_cat,batch_size=32,nb_epoch=2,verbose=1)
Epoch 2/2
90000/90000 [==============================] - 912s - loss: 0.7580 - acc: 0.680
score=model3.evaluate(X_test_tfidf_clean_bigram.toarray(),y_test_cat,verbose=1)
score[1]*100
 Out[224]: 61.860000000000007
vectorizer = CountVectorizer(analyzer = "word",  tokenizer = None, ngram_range=(1,3),
preprocessor = None, stop_words = None, max_features = 10000)
model3.
fit(X_train_tfidf_clean_bigram.toarray(),y_train_cat,batch_size=32,nb_epoch=2,verbose=1)
Epoch 2/2
90000/90000 [==============================] - 588s - loss: 0.7892 - acc: 0.664
score=model3.evaluate(X_test_tfidf_clean_bigram.toarray(),y_test_cat,verbose=1)
score[1]*100

 Out[237]: 62.129999999999995
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
t1=pd.DataFrame(clean_X_train)
t2=pd.DataFrame(clean_X_test)
X_train_lemmatize=t1.text.apply(wordnet_lemmatizer.lemmatize)
X_test_lemmatize=t2.text.apply(wordnet_lemmatizer.lemmatize)
vectorizer = CountVectorizer(analyzer = "word",   tokenizer = None, ngram_range=(1,2),
model3.add(Dense(748,input_dim=5000,init="normal",activation="relu"))
model3.add(Dropout(0.7))
model3.add(Dense(5,init="normal",activation="softmax"))
model3.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
mod-
el3.fit(X_train_tfidf_clean_bigram.toarray(),y_train_cat,batch_size=32,nb_epoch=2,verbose=1)
Epoch 2/2
90000/90000 [==============================] - 296s - loss: 0.8347 - acc: 0.641
score=model3.evaluate(X_test_tfidf_clean_bigram.toarray(),y_test_cat,verbose=1)
score[1]*100
 Out[273]: 61.350000000000001
from nltk.stem import SnowballStemmer
snowball_stemmer = SnowballStemmer("english")
t1=pd.DataFrame(clean_X_train)
t2=pd.DataFrame(clean_X_test)
X_train_stem=t1.text.apply(snowball_stemmer.stem)
X_test_stem=t2.text.apply(snowball_stemmer.stem)
vectorizer = CountVectorizer(analyzer = "word",   tokenizer = None, ngram_range=(1,2),
```

```
model3.
fit(X_train_tfidf_clean_bigram.toarray(),y_train_cat,batch_size=32,nb_epoch=2,verbose=1)
Epoch 2/2
90000/90000 [==============================] - 306s - loss: 0.8355 - acc: 0.639
score=model3.evaluate(X_test_tfidf_clean_bigram.toarray(),y_test_cat,verbose=1)
score[1]*100
Out[44]: 61.0
```

Tree-based models

```
from sklearn import tree
parameters_grid = {
    'criterion':['gini','entropy'],
    'max_depth':k,
    'max_features':['sqrt',None],
    'min_samples_leaf':[1,5,10,15,20,25,30,40,50],
    'min_impurity_split':[0.1,0.05,0.01,0.005,0.001,0.0001,0.00001,0.000001,0]
    }
from sklearn.model_selection import StratifiedShuffleSplit
cv = StratifiedShuffleSplit(n_splits = 3,test_size = 0.2, random_state = 0)
from sklearn import grid_search
randomized_grid_cv = grid_search.RandomizedSearchCV(clf, parameters_grid, scoring = 'accu-
racy', cv = list(cv.split(X_train_tfidf_clean_bigram,y_train)), n_iter = 50, random_state = 0)
randomized_grid_cv.fit(X_train_tfidf_clean_bigram,y_train)
randomized_grid_cv.best_estimator_
Out[84]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=57,
                    max_features=None, max_leaf_nodes=None,
                    min_impurity_split=1e-06, min_samples_leaf=30,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    presort=False, random_state=0, splitter='best')
randomized_grid_cv.best_score_
Out[85]: 0.44790264692687304
from sklearn.ensemble import RandomForestClassifier
parameters_grid2 = {
    'n_estimators':[10,30,50,80,100,150,200,300,400,500]
    }
clf=RandomForestClassifier(random_state=1234)
grid_search_cv=grid_search.GridSearchCV(clf, parameters_grid2, scoring = 'accuracy', cv =
list(cv.split(X_train_tfidf_clean_bigram,y_train)), n_jobs = 4)
grid_search_cv.fit(X_train_tfidf_clean_bigram,np.ravel(y_train))
clf=RandomForestClassifier(n_estimators=250,random_state=1234)
k=[0.01,0.015,0.02,0.025,0.03,0.03,0.1,0.2,0.3,0.5,0.7]
parameters_grid2 = {
    'max_features':k
    }
grid_search_cv=grid_search.GridSearchCV(clf, parameters_grid2, scoring = 'accuracy', cv =
list(cv.split(X_train_tfidf_clean_bigram,y_train)), n_jobs = 4)
grid_search_cv.fit(X_train_tfidf_clean_bigram,np.ravel(y_train))
mean_arr= list(grid_search_cv.grid_scores_[i][1] for i in
accuracy_score(y_test,prediction)
Out[240]: 0.54350898445386631
from sklearn.decomposition import TruncatedSVD
```

```
svd = TruncatedSVD(n_components=100,algorithm="arpack", n_iter=7, random_state=42)
X_train_svd=svd.fit_transform(X_train_tfidf_clean_bigram)
X_test_svd=svd.transform(X_test_tfidf_clean_bigram)
clf=RandomForestClassifier(n_estimators=300,random_state=1234,n_jobs=4)
clf.fit(X_train_svd,np.ravel(y_train))
prediction=clf.predict(X_train_svd)
print('точность на тренировочной выборке:')
print(accuracy_score(y_train,prediction))
print('**********'*5)
prediction=clf.predict(X_test_svd)
print('точность на тестовой выборке:')
print(accuracy_score(y_test,prediction))
```

```
точность на тренировочной выборке:
0.999629853961
**************************************************
точность на тестовой выборке:
0.519348542971
```

```
clf=RandomForestClassifier(n_estimators=200,random_state=1234,max_depth=15,min_samples
_leaf=10,min_impurity_split=1e-02,n_jobs=4)
clf.fit(X_train_svd,np.ravel(y_train))
prediction=clf.predict(X_train_svd)
print('точность на тренировочной выборке:')
print(accuracy_score(y_train,prediction))
print('**********'*5)
prediction=clf.predict(X_test_svd)
print('точность на тестовой выборке:')
print(accuracy_score(y_test,prediction))
```

```
точность на тренировочной выборке:
0.810720775288
**************************************************
точность на тестовой выборке:
0.511407227943
```

```
parameters_grid2 = {
   'max_features':[0.05,0.1,0.15,0.2],
   'n_estimators':[100,150,200,300,500],
   'min_samples_leaf':[1,5,10,15,20],
   'max_depth':[10,15,20,30,50],
   'min_impurity_split':[1e-02,1e-03,1e-07]
   }
clf=RandomForestClassifier(random_state=1234)
rs=grid_search.RandomizedSearchCV(clf, parameters_grid2, scoring = 'accuracy', cv =
list(cv.split(X_train_svd,y_train)), n_jobs = 4)
rs.fit(X_train_svd,np.ravel(y_train))
rs.best_estimator_
```

```
Out[326]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=50, max_features=0.2, max_leaf_nodes=None,
                       min_impurity_split=0.01, min_samples_leaf=5,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=150, n_jobs=1, oob_score=False, random_state=1234,
                       verbose=0, warm_start=False)
```

```
prediction=rs.predict(X_train_svd)
print('точность на тренировочной выборке:')
print(accuracy_score(y_train,prediction))
```

```
print('**********'*5)
prediction=rs.predict(X_test_svd)
print('точность на тестовой выборке:')
print(accuracy_score(y_test,prediction))
```

```
точность на тренировочной выборке:
0.966350360051
**************************************************
точность на тестовой выборке:
0.520156134329
```

```
from sklearn.ensemble import ExtraTreesClassifier
clf=ExtraTreesClassifier(max_depth=50,max_features=0.2, max_leaf_nodes=None,
        min_impurity_split=0.01, min_samples_leaf=5,
        min_samples_split=2, min_weight_fraction_leaf=0.0,
        n_estimators=150,n_jobs=4)
clf.fit(X_train_svd,np.ravel(y_train))
prediction=clf.predict(X_train_svd)
print('точность на тренировочной выборке:')
print(accuracy_score(y_train,prediction))
print('**********'*5)
prediction=clf.predict(X_test_svd)
print('точность на тестовой выборке:')
print(accuracy_score(y_test,prediction))
```

```
точность на тренировочной выборке:
0.983461201965
**************************************************
точность на тестовой выборке:
0.508849855307
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=100)
X_train_pca=pca.fit_transform(X_train_tfidf_clean_bigram.toarray())
X_test_pca=pca.transform(X_test_tfidf_clean_bigram.toarray())
clf=RandomForestClassifier(max_depth=50,max_features=0.2, max_leaf_nodes=None,
        min_impurity_split=0.01, min_samples_leaf=5,
        min_samples_split=2, min_weight_fraction_leaf=0.0,
        n_estimators=150,n_jobs=4)
clf.fit(X_train_pca,np.ravel(y_train))
prediction=clf.predict(X_train_pca)
print('точность на тренировочной выборке:')
print(accuracy_score(y_train,prediction))
print('**********'*5)
prediction=clf.predict(X_test_pca)
print('точность на тестовой выборке:')
print(accuracy_score(y_test,prediction))
```

```
точность на тренировочной выборке:
0.968150615788
**********
точность на тестовой выборке:
0.522511609126
```

```
from sklearn.ensemble import GradientBoostingClassifier
clf=GradientBoostingClassifier()
parameters_grid2 = {
   'n_estimators':[50,100,150,200,300,500],
```

```
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2,0.3],
    'max_depth':[2,3,4,5,6,10],
    'min_samples_leaf':[1,2,3,4,5,10],
    'max_features':[5,10,15,20]
    }
rs=grid_search.RandomizedSearchCV(clf, parameters_grid2, scoring = 'accuracy', cv =
list(cv.split(X_train_pca,y_train)), n_jobs = 4)
rs.fit(X_train_pca,np.ravel(y_train))
prediction=rs.predict(X_train_pca)
print('точность на тренировочной выборке:')
print(accuracy_score(y_train,prediction))
print('**********'*5)
prediction=rs.predict(X_test_pca)
print('точность на тестовой выборке:')
print(accuracy_score(y_test,prediction))
```

```
точность на тренировочной выборке:
0.609866074433
**************************************
точность на тестовой выборке:
0.552055993001
```

```
clf=GradientBoostingClassifier(n_estimators=200, max_depth=5,min_samples_leaf=5)
clf = clf.fit(X_train_tfidf_clean_bigram,np.ravel(y_train))
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]
dict_pd=pd.DataFrame(vectorizer.get_feature_names(),columns=['Words'])
indeces20=dict_pd.iloc[[1686, 2551,  962, 1631, 3818,  315, 1420, 1241,   93,  368, 2631,
       211, 2210, 1319, 2329, 2829, 4444,  930, 4924, 2896]]
indeces20
```

	Words
1686	great
2551	not
962	delicious
1631	good
3818	symbolfornumber
315	best
1420	food
1241	excellent
93	amazing
368	bland

```
tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')

def review_to_sentences( review, tokenizer):
    raw_sentences = tokenizer.tokenize(review.strip())
    sentences = []
    for raw_sentence in raw_sentences:
        if len(raw_sentence) > 0:
            sentences.append( review_to_wordlist(raw_sentence))
    return sentences
sentences = []
```

```
print ("Парсим наши отзывы")
for review in X_train["text"]:
    sentences += review_to_sentences(review, tokenizer)
num_features = 300
min_word_count = 30
num_workers = 4
context = 10
downsampling = 1e-3
from gensim.models import word2vec
model = word2vec.Word2Vec(sentences,sg=1, workers=num_workers,size=num_features,
min_count = min_word_count,window = context, sample = downsampling)
```

```
model.most_similar("tasty")
```

```
[('delicious', 0.6682024002075195),
 ('yummy', 0.6599723100662231),
 ('flavorful', 0.6180805563926697),
 ('delish', 0.5896403193473816),
 ('good', 0.5753912329673767),
 ('tastey', 0.5711269378662109),
 ('satisfying', 0.5399523973464966),
 ('scrumptious', 0.5279601812362671),
 ('deliciously', 0.5269473195075989),
 ('accompaniment', 0.5182754993438721)]
```

```
model.most_similar("croissant")
```

```
[('chocolat', 0.7659599781036377),
 ('croissants', 0.7565087080001831),
 ('raisin', 0.7328780889511108),
 ('aux', 0.7313952445983887),
 ('nutella', 0.6807685494422913),
 ('challah', 0.6772316694259644),
 ('hotcake', 0.6657204627990723),
 ('eclair', 0.6573852300643921),
 ('quiche', 0.6558743715286255),
 ('almond', 0.6526625156402588)]
```

```
model.most_similar_cosmul(positive=['italy', 'croissant'], negative=['pizza'])
```

```
[('france', 0.9873804450035095),
 ('chocolat', 0.9546630382537842),
 ('paris', 0.9524206519126892),
 ('croissants', 0.952370285987854),
 ('parisian', 0.9442649483680725),
 ('aux', 0.9347772002220154),
 ('tartine', 0.9069862365722656),
 ('raisin', 0.8985605835914612),
 ('jules', 0.8976535797119141),
 ('eclair', 0.8920069336891174)]
```

```
list_words=["pizza","pepperoni","neapolitan","calzone","margherita","croissant","chocolat","nut
ella","hotcake","eclair","tasty",
        "delicious","yummy"]
color_map=[1,1,1,1,1,2,2,2,2,2,3,3,3]
word2wec1=np.zeros((13,300),dtype="float32")
k=0
```

```
for i in list_words:
    word2wec1[k]=model[i]
    k=k+1
from sklearn.manifold import TSNE
model_tsne = TSNE(n_components=2, random_state=0)
np.set_printoptions(suppress=True)
word2wec1_tsne=model_tsne.fit_transform(word2wec1*1000000)
x=list(range(13))
for i in range(0,len(word2wec1_tsne)):
    x[i]=word2wec1_tsne[i][0]
y=list(range(13))
for i in range(0,len(word2wec1_tsne)):
    y[i]=word2wec1_tsne[i][1]
fig, ax = plt.subplots(figsize=(10, 10))
ax.scatter(x, y,c=color_map,cmap="Set1",s=150)

for i, txt in enumerate(list_words):
    ax.annotate(txt, (x[i],y[i]),size=15)
```


```
def makeFeatureVec(words, model, num_features):
    featureVec = np.zeros((520,num_features),dtype="int32")
    counter=0
    index2word_set = set(model.wv.index2word)
    i=0
    for word in words:
        if word in index2word_set:
            featureVec[i]= model[word]*1000000000000
            i=i+1
    return featureVec
def getAvgFeatureVecs(reviews, model, num_features):
    reviewFeatureVecs = np.zeros((len(reviews),520,num_features),dtype="float32")
        counter = 0.
        for review in reviews:
            reviewFeatureVecs[counter] = makeFeatureVec(review, model,num_features)
            counter = counter + 1.
    return reviewFeatureVecs
```

```
print ("Делаем эмбединг для тренировочной выборки")
clean_train_reviews = []
for review in X_train.text[0:20000]:
    clean_train_reviews.append( review_to_wordlist(review))
trainDataVecs = getAvgFeatureVecs( clean_train_reviews, model, num_features )
trainDataVecs = trainDataVecs.reshape(trainDataVecs.shape[0],520, 300,1)
trainDataVecs2 = trainDataVecs.reshape(trainDataVecs.shape[0],520, 300,1)
model_cnn = Sequential()
        model_cnn.add(Convolution2D(32,200, 3, activation='relu', input_shape=(520,200,1)))
        model_cnn.add(GlobalMaxPooling2D())
        model_cnn.add(Dropout(0.25))
        model_cnn.add(Dense(128, activation='relu'))
        model_cnn.add(Dropout(0.5))
        model_cnn.add(Dense(5, activation='softmax'))
        model_cnn.compile(loss='categorical_crossentropy', optimiz-
er='RMSprop',metrics=['accuracy'])
        model_cnn.fit(trainDataVecs[0:10000], Y_train[0:10000], batch_size=16, nb_epoch=1,
verbose=1)
 Epoch 1/1
 10000/10000 [==============================] - 6793s - loss: 11.8468 - acc: 0.2650

with tf.device('/cpu:0'):
        model_cnn = Sequential()
        model_cnn.add(Convolution1D(256,300, activation='relu', input_shape=(300,1)))
        model_cnn.add(GlobalMaxPooling1D())
        model_cnn.add(Dense(128, activation='relu'))
        model_cnn.add(Dropout(0.25))
        model_cnn.add(Dense(5, activation='softmax'))
        model_cnn.compile(loss='categorical_crossentropy', optimiz-
er='Adam',metrics=['accuracy'])
        model_cnn.fit(trainDataVecs2, Y_train, batch_size=32, nb_epoch=10, verbose=1)
Epoch 10/10
59436/59436 [==============================] - 8s - loss: 0.9153 - acc: 0.5925

score = model_cnn.evaluate(testDataVecs2, Y_test, batch_size=32,verbose=0)
print(score[1]*100)

58.65805236
from sklearn import linear_model
logreg=linear_model.LogisticRegression(C=1e5)
logreg=logreg.fit(trainDataVecs,np.ravel(y_train))
prediction_test=logreg.predict(testDataVecs)
accuracy_score(np.ravel(y_test), prediction_test)
X_train_tok = [x.split() for x in x_pos['text']]
dictionary = corpora.Dictionary(X_train_tok)
doc_term_matrix = [dictionary.doc2bow(doc) for doc in X_train_tok]
ldamodel = LdaModel(doc_term_matrix, num_topics=15, id2word = diction-
ary,alpha='auto',passes=30)
```

```
ldamodel.print_topics(num_topics=15, num_words=10)
```

```
[(0,
  '0.019*"bar" + 0.017*"old" + 0.010*"music" + 0.009*"cool" + 0.007*"whiskey" + 0.007*"game" + 0.007*"crowd" + 0.007*"playing"
 + 0.006*"play" + 0.006*"bathroom"'),
 (1,
  '0.022*"like" + 0.015*"good" + 0.014*"place" + 0.011*"little" + 0.009*"order" + 0.009*"pretty" + 0.009*"eat" + 0.008*"make" +
0.008*"even" + 0.008*"always"'),
 (2,
  '0.040*"coffee" + 0.022*"shop" + 0.013*"bakery" + 0.011*"almond" + 0.010*"pastries" + 0.010*"la" + 0.009*"store" + 0.009*"coo
kies" + 0.009*"polish" + 0.008*"deli"'),
 (3,
  '0.024*"chef" + 0.010*"meats" + 0.010*"salt" + 0.009*"ingredients" + 0.007*"menu" + 0.006*"flavors" + 0.006*"duck" + 0.005*"t
asting" + 0.004*"legume" + 0.004*"dishes"'),
 (4,
  '0.035*"bar" + 0.028*"drinks" + 0.021*"wine" + 0.020*"restaurant" + 0.020*"dining" + 0.016*"drink" + 0.015*"tables" + 0.015
*"seating" + 0.014*"menu" + 0.014*"small"'),
 (5,
  '0.070*"thai" + 0.039*"curry" + 0.029*"spicy" + 0.024*"chicken" + 0.021*"pad" + 0.021*"vegan" + 0.021*"spice" + 0.019*"buffe
t" + 0.019*"indian" + 0.017*"tea"'),
 (6,
  '0.044*"sushi" + 0.025*"soup" + 0.022*"roll" + 0.022*"chinese" + 0.021*"fish" + 0.019*"spicy" + 0.019*"rolls" + 0.016*"rice"
 + 0.015*"noodles" + 0.014*"chicken"'),
 (7,
  '0.061*"tacos" + 0.031*"taco" + 0.027*"mexican" + 0.024*"chips" + 0.023*"salsa" + 0.019*"dog" + 0.019*"pho" + 0.017*"burrito"
+ 0.014*"hot" + 0.013*"chicken"'),
 (8,
  '0.102*"beer" + 0.041*"selection" + 0.037*"bar" + 0.035*"beers" + 0.024*"hour" + 0.022*"happy" + 0.014*"tap" + 0.012*"craft"
 + 0.011*"game" + 0.011*"hummus"'),
 (9,
  '0.019*"salad" + 0.015*"sauce" + 0.012*"bread" + 0.010*"cooked" + 0.010*"dessert" + 0.010*"cheese" + 0.009*"perfectly" + 0.00
9*"flavor" + 0.009*"sweet" + 0.009*"delicious"'),
 (10,
  '0.072*"food" + 0.059*"great" + 0.044*"good" + 0.038*"place" + 0.031*"service" + 0.026*"best" + 0.024*"delicious" + 0.021*"pi
ttsburgh" + 0.017*"amazing" + 0.017*"definitely"'),
 (11,
  '0.131*"pizza" + 0.074*"burger" + 0.033*"fries" + 0.032*"cheese" + 0.025*"burgers" + 0.020*"crust" + 0.016*"toppings" + 0.014
*"sauce" + 0.013*"pizzas" + 0.010*"slice"'),
 (12,
  '0.058*"sandwich" + 0.042*"breakfast" + 0.038*"brunch" + 0.028*"fries" + 0.018*"eggs" + 0.018*"french" + 0.016*"pancakes" +
 0.015*"toast" + 0.014*"coffee" + 0.013*"bacon"'),
 (13,
  '0.020*"time" + 0.018*"first" + 0.016*"back" + 0.013*"night" + 0.013*"made" + 0.012*"next" + 0.012*"table" + 0.010*"friend" +
0.010*"last" + 0.009*"enjoyed"'),
 (14,
  '0.049*"chicken" + 0.047*"cheese" + 0.036*"wings" + 0.030*"mac" + 0.022*"bbq" + 0.021*"pork" + 0.021*"sauce" + 0.017*"sandwic
h" + 0.017*"n" + 0.015*"fried"')]
```

```
model3=Sequential()
model3.add(Dense(748,input_dim=7000,init="normal",activation="relu"))
# model.add(LSTM(200,dropout_W=0.2,dropout_U=0.2))
model3.add(Dropout(0.7))
model3.add(Dense(3,init="normal",activation="softmax"))
model3.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
model3.fit(X_train_tfidf_clean_trigram.toarray(),y_train_cat,batch_size=32,nb_epoch=2,verbose=1)
```

```
Epoch 1/2
59436/59436 [==============================] - 79s - loss: 0.4804 - acc: 0.8043
Epoch 2/2
59436/59436 [==============================] - 76s - loss: 0.3609 - acc: 0.8562

<keras.callbacks.History at 0x18cde056f98>
```

```
score=model3.evaluate(X_test_tfidf_clean_trigram.toarray(),y_test_cat,verbose=1)
score[1]*100
```

```
14752/14859 [============================>.] - ETA: 0s

83.282858875415727
```

Приложение 3. Облака слов.

Рисунок 1П - облако слов для отзывов с оценкой 1

Рисунок 2П - облако слов для отзывов с оценкой 2

Рисунок 3П - облако слов для отзывов с оценкой 3

Рисунок 4П - облако слов для отзывов с оценкой 4

Рисунок 5П - облако слов для отзывов с оценкой 5

Приложение 4. Опрос.

В проведенном опросе было предложено оценить следующие отзывы.

Отзыв 1.

Отличный сервис и еда, и самый большой плюс - это отсутствие платы за свою еду. Атмосфера уютная, а в хорошую погоду можно сидеть на свежем воздухе. Еда, в общем, была хорошей. На завтрак я взял пурпурные равиоли с картофелем, увенчанные сливочным соусом песто. Начинка была немного безвкусной, и сливочная природа песто заполняла собой все блюдо. Моя закуска была удивительной - жареные оладьи из киноа. Мое главное блюдо состояло из запеченных гребешков, ко-

торые были приготовлены очень хорошо. Подавались они с пастой феттучини с горошком и сливочным соусом. Блюда были такие большие, что мне пришлось поделиться с другом. Несмотря на одно хорошее блюдо, мы обязательно вернемся и попробуем другие блюда. Здесь очень большой выбор для вегетарианцев.

Отзыв 2

Я действительно люблю все, что я ем здесь, но меня всегда пугает парень за прилавком, который кажется таким злым все время. Но еда действительно вкусная. Я часто ем шаверму с курицей и салат из шпината и свеклы на обед. Хорошее комбо. Кроме того, хотя трудно конкурировать с Salem- там фалафель, вероятно, лучший. У них есть эти удивительные маленькие кукурузные снеки, которые просто ... ням

Отзыв 3

Горячие пирожки - лучшее, что я когда-либо ел, и картофель с соусом хорош. Клубничные пирожки - то, что сделало их знаменитыми. За 7 баксов вы получаете 2 щедрые горсти клубники, коричневого сахара и крема. Утром там очереди, но сервис по-прежнему эффективен и прекрасен.

Отзыв 4

3.5 / 5 -Пастицсио: восхитительное блюдо с прекрасным ароматом и пропитанное соусом бешамель

2.0 / 5 - Гиро: очень маленькие порции, не имеющие вкуса

3,5 / 5 - Ламб : восхитительный, но маленький

Отзыв 5

Я могу написать этот обзор только из-за их бабл чая. Это один из, если не самый лучший бабл чай, который я пробовал в Питтсбурге. У них он получается во всех вкусах, а мой любимый - их молочный кофе

Отзыв 6

Я приезжаю сюда регулярно почти 30 лет, в основном с самого рождения. Некоторые старшие официантки уже как часть семьи в этот момент, так что я могу быть немного предвзятым. Еда всегда хороша, официанты дружелюбны, и повара развлекают, не раздражая, как во многих других местах с хибачи(японская печь). Я не большой любитель суши, но то, что я пробовал, было хорошо. У них есть фантастический выбор вин, пива и сакэ. Я, вероятно, буду ездить сюда еще 30 лет

Отзыв 7

Лично знаю 6 людей, которые отравились гамбургерами и / или курицей, которые подаются в Urban Tap. Ел там прошлой ночью с друзьями, и двух девочек, которые заказали гамбургеры, тошнит нон-стоп, все утро.

Отзыв 8

Хотя мой блюдо на гриле был в порядке, я не мог это не написать. Я ожидал большего вкуса. 2 звезды убираю из-за УЖАСНОГО обслуживания. Я получил это - жирная ложка, обращение на сленге, и т. Д. Они потратили 45 минут, чтобы сделать яйца? Глупая официантка сказала, что они немного заняты. Ну камон. Мы были там слишком долго. Это как-то испортило остаток дня. Это не лучший завтрак в Питтсбурге в любом случае. Это было прекрасно, потому что это было, но ужасное обслуживание = не возвращаться. Во время моих поездок в Питтсбург мне стоит поискать другое место для завтрака.

Отзыв 9

Пришел сюда сегодня впервые, и, вероятно, не вернусь, обслуживание было медленным, в общей сложности час. Не тратьте свое время

Отзыв 10

Я ел и здесь и не здесь в течение большей части двух последних десятилетий. В еде нет ничего особенного, но она вполне хороша. Кофе всегда свежий, и Кровавая Мери хороша. Пойдет, если угодно.

Правильные ответы: 4,4,4,3,5,1,2,2,3

Список таблиц

Список рисунков